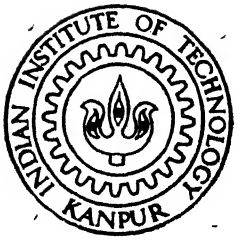
DISEMP : A Processor Design for Machine Translation Applications

by
M. R. WARSI



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JANUARY 1997

DISEMP : A Processor Design for Machine Translation Applications

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

M. R. Warsi

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

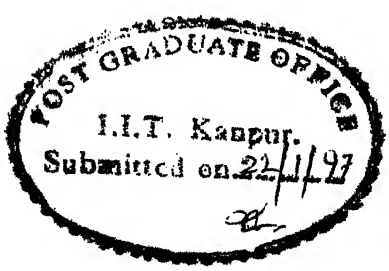
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

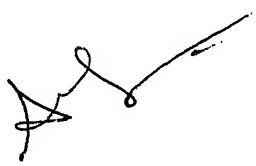
January 1997

CSE-1997-M-WAR-DIS

CERTIFICATE

This is to certify that the work contained in the thesis entitled DISEMP : A Processor Design for Machine Translation Applications by M. R. Warsi has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.





Dr. Ajai Jain,
Associate Professor,
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur.

Abstract

Hardware support to machine translation (MT) is needed due to large processing requirement in sentence translation. In HEBMT (Hybrid Example Based Machine Translation) approach for MT, translation is carried out in three major steps -

1. Analyse sentence to retrieve word's characteristics
2. Determine sentence pattern
3. Match pattern with examples

For MT applications, dictionary search and example matching are basic operations and they take most of the time of translation. In present thesis, design of a processor, for dictionary search and example matching, is presented. Processor is named as DISEMP (DIctionary Search and Example Matching Processor).

DISEMP accepts a word or a sentence pattern as input. The word is searched in the dictionary and it's characteristics is passed as the result. For a sentence pattern, distances are calculated for all examples. The target pattern of the example, which has least distance, is passed as result. DISEMP design has been made flexible so that, it can perform complicated operations such as substring matching. DISEMP design, also, incorporates some performance enhancement features. It supports partitioning of database to narrow down search and match space. Cache, in DISEMP, makes use of repetition of words in documents.

Acknowledgements

I am grateful to my thesis supervisor, Dr. A. Jain, for giving me an interesting problem to work on. He guided and motivated me for exploring novel solutions. Because of his continuous encouragements, I never lost confidence during work.

Special thanks are due to my friends Deepak, Utpal, Ramesh, Milind, Bhole and others of M.Tech95 batch who tolerated my long absences from their midst. I must thank them for listening to my problems and for helping me in overcoming them.

I am grateful to Dr. H. M. Srivastava and faculty members of K. N. I. T. Sultanpur for providing me support in carrying out study at IIT, Kanpur.

Last, but not the least, I am grateful to my parents and brothers Tahsin and Zia for tolerating my absence from home for a long time. I must thank them for corresponding and encouraging me.

Contents

Chapter 1

INTRODUCTION

Technology has been evolving rapidaly to bring people close together. Research. in area of networking and communication, has helped a lot in bridging the gap. Crossing the barrier of language will have far reaching impact in this direction. Lot of information exist in different languages which need to be shared among people. That's why, machine translation has been a topic of interest for researchers over a long time.

1.1 What is Machine Translation (MT)

Machine Translation (MT) means adding capability of translation, from one language to another, to computer. For example, it may take a document in Hindi and may produce translated document in English. From application point of view, MT has lots of potential. Rich information exists in each language which can be further classified into scientific information, literary information, legal information etc. Making information available in one language to another would bring about another information revolution. MT is an attempt in that direction.

Inspite of good efforts put into this area, lot of work is yet to be done to achieve perfect translation. The difficulty of machine translation lies in the ambiguity present in natural languages. Words and sentences have different meaning in different context which makes the task of translation difficult. Human beings,

because of their knowledge and experience, are able to resolve ambiguity. But very limited success has been achieved in creating the same knowledge in machines.

1.2 Approaches

Traditional approaches to machine translation can be broadly classified into three categories [Jai95] -

- Direct Translation Strategy,
- Transfer Strategy,
- Interlingua Strategy

Some of these approaches are rule driven and they require rules to be handcrafted by experts and put into the systems. Handcrafting of rules is a difficult process and satisfactory translation is not obtained because of it being incomplete and inconsistent. Another approach, known as example based approach, has been proposed which doesn't require formulation of rules. The example based approach stores numerous examples in database and chooses the appropriate example for translation. In HEBMT (Hybrid Example Based Machine Translation) approach [Jai95], examples are stored in abstracted form. This approach has the advantage of achieving satisfactory translation with lesser disk space requirement. A software, using this approach, has been developed by R. Jain et. al. In this approach, dictionary search and example matching operations take most of the time of translation. Speed of translation can be, further, improved by performing these operations in hardware.

1.3 Hardware Support

Real time solution to MT application is difficult through software. It takes lot of time, in software, to search into the dictionary base and to match examples. Few architectures, for machine translation systems, have been proposed [H.K93b,

H.K93a, Suk96]. Kitano proposed an architecture for MBMT (Memory Based Machine Translation) system. Another machine translation system was proposed by P. Sukumar [Suk96]. The design of this system was based on hybrid example based approach. Sukumar's architecture consists of specialised processor (named as DiSe and MediCal) to perform dictionary search and example matching. DiSe and MediCal offered limited flexibility as they were instructionless.

1.4 The Present Work

Present work has been carried out to design a processor for machine translation (MT) application. To overcome the limitation of Sukumar's processor, flexibility has been an important consideration during design. The work has been carried out in three phases -

1. Performance Measurement of A MT Software
2. Design and Simulation of Processor
3. Design Verification

In the first phase, a MT software was studied to identify potential areas for hardware. Study indicates that dictionary search and example matching take most of the time of translation, and a processor for them will reduce translation time. In the second phase, a processor (DISEMP) was designed and simulated. DISEMP is an abbreviation for DIctionary Search and Example Matching Processor. DISEMP accepts a word and searches it in the dictionary. The dictionary contains the words and their characteristics. If the word is found in the dictionary, it's characteristic is passed as the result. DISEMP, also, accepts a sentence pattern and matches it with examples. The example base contains examples and their target language pattern. In an example matching operation, distances are calculated for all examples. Target pattern of the example, having least distance, is passed as the result of example matching. The dictionary base and the example base are partitioned to narrow down search and match space. The DISEMP design is flexible so that, it can perform

complicated operations such as substring match. In the third phase, design was verified by integrating it with MT software. In this phase, MT software used the services of DISEMP to carry out the translation. Correct translation, carried out by the MT software, verified the DISEMP design.

1.5 Thesis Organisation

Thesis has been organised into eight chapters. Chapter 2 describes the hybrid example based approach for machine translation. The chapter also gives an overview of potential hardware schemes for machine translation. Chapter 3 discusses the design issues, considerations and the approach taken to design the processor. The chapter 4 describes about the external environment of DISEMP. In chapter 5, DISEMP design has been discussed. The chapter 6 describes the internal operations of DISEMP in detail. Implementation details are presented in chapter 7. And finally, chapter 8 concludes the thesis.

Chapter 2

HARDWARE AND SOFTWARE OVERVIEW

Early attempts in MT were made using rule driven approach. Difficulty in formulation of rules, led to evolution of example based approach [RJR95] in late eighties. In this approach, raw examples and their translation in target language, are stored in example base. During the process of translation, a raw example, which matches closely with input sentence, is selected and associated translated sentence, in target language, is used for translating the input sentence. Storing the examples, in raw form, not only occupies lot of disk space, but matching the input sentence with examples also takes time. The HEBMT (Hybrid Example Based Machine Translation) approach [Jai95], derived from example based approach, stores the examples in abstracted form. Abstraction of raw examples results in disk space saving and also reduces the time to match abstracted examples with input sentence. Translation is carried out by first analysing the input sentence, which involves dictionary search, and then matching the input sentence with abstracted examples. Dictionary search and example matching take most of the time of translation process. Performance, of the HEBMT system, can be further improved by performing these operations in hardware. Dictionary search and example matching operations are basic operations for MT application, and different MT systems can be developed using them.

This chapter first describes the HEBMT approach, and later presents various hardware schemes which have been proposed for string searching and pattern matching.

2.1 The HEBMT Approach

In this approach, translation is carried out in three major steps - morphological analysis (MA), identification of syntactic groups (ISG) and example matching (EM). MA phase analyses the words of a sentence and retrieves their characteristics from dictionary. ISG phase forms group of words, known as syntactic unit, such as *John's brother*, *very good work* etc. And finally, example matching phase chooses the best example based on which translation is carried out. For example, in order to translate following sentence -

vaha shyam ke ghar ja raha he

MA phase identifies root word as *jana*, tense of sentence as *present continuous* and gender as *male*. Morph analysis further retrieves the characteristics of other words such as *vaha* is pronoun, *shyam* is proper noun and *ghar* is noun. This information is fed to the ISG unit which groups the words as below -

(*vaha*) (*shyam ke ghar*) (*ja raha he*)

With each syntactic unit, few attributes are attached which could be gender, person, number etc. depending on it's type. Above sentence can be abstracted to following form -

$$np1[M, S, A],\ np2[M, S, I],\ mv[T]$$

where $np1$ (noun phrase of type 1), $np2$ (noun phrase of type 2) and mv (verb phrase) are types of syntactic unit. Values within square bracket are attributes of syntactic unit (M for *Male*, S for *Single*, A for *Animate* and T for *Transitive*). Source sentence pattern, in above form, is given to the example matching unit which matches above pattern against examples in example base. Examples are also stored

in similar patterns. With each example in example base, there is a target pattern associated. Target pattern for the above translation is

$$np1\ mv\ \{to\}\ np2$$

This target pattern will be giving the translation as -

he is going to shyam's house

2.1.1 Morphological Analysis (MA)

Morphological analysis determines root word from it's derivative and retrieves information associated with it. Information, associated with a root word, depends on the type of word and it could be number, person, gender, target language meaning, semantic tag etc. Problems, in morph analysis, are irregularity and historicity in derivatives. For example, the word *destroy* has corresponding noun as *destruction*, while noun form of *employ* is not *empluction*. Two suggested implementation for morph analysis are - 1) through paradigm file and 2) through suffix file. There is a paradigm file for each part of speech. Prefixes of the word are extracted and searched in the paradigm file. In suffix file strategy, suffixes of the derivatives are stored in the suffix file. Derivative's suffixes are matched with that in file and root word candidates are obtained which are searched into the dictionary.

Morph analysis is carried out separately for verb part and remaining part of the sentence. Verb part analysis returns root verb of a sentence, number and gender of subject of the sentence, tense of verb, and corresponding verb form in target language. In Hindi, verb can be classified as simple verb, complex verb (*noun/adj* + *kara/ho/de* such as *seva karana*) or compound verb (*simple verb* + *simple verb* such as *le jana*). Morph analysis looks for all of them.

Analysis of remaining part of the sentence is carried out to determine root word of noun, pronoun or adjective. Root word is identified by joining the words, by breaking the word or by applying the post-position. For example, the root word of *kal purja* appears as two words in sentence, so they are joined and searched.

2.1.2 Identification of Syntactic Group (ISG)

This phase groups the words to form a syntactic unit. Abstracted form of sentence is represented by collection of syntactic units. [Jai95] has implemented this phase by using finite state machine (FSM). Syntactic units can move freely in a sentence in Hindi. FSM takes syntactic and semantic information of words from morph analyser and generates expectations for next word. If type of next word is one of generated expectations, FSM continues till it reaches the final state otherwise it halts and reports error. For example, if word *mera* has been encountered, which is a possessive case, expectation for next word could be adverb, adjective, common noun etc. This phase also translates the syntactic units into target language form.

2.1.3 Example Matching (EM)

ISG phase transforms the sentence into abstracted form which goes as input to example matching phase. Abstracted pattern is matched against the examples in example base in this phase. Abstracted pattern contains syntactic units and their attribute information. Different types of syntactic units have different types of attribute information associated with them. For example, a noun phrase may have attributes as gender, number, person etc. while attribute of a verb phrase, is whether it's transitive or intransitive.

Syntactic units may also have semantic tags attached to them which can be used to resolve ambiguity in the sentence. A word in a sentence may have multiple meaning in target language and appropriate meaning is chosen by determining the context in which the word has been used. Context information is added through semantic tags. For example, word *dabav* in Hindi may have meaning in English as *pressure*, *stress* etc. and appropriate meaning is chosen by looking at the context in which word *dabav* has been used. Semantic information is classified in the form of a tree as shown in Fig 2.1. Semantic tags, of input sentence syntactic units, and that of example sentence are compared against semantic tree. More they are closer to their common ancestors, better is the example for translation.

The example base is partitioned to speed up the matching process. One partitioning scheme [Jai95] partitions it at several level where first level of partitioning

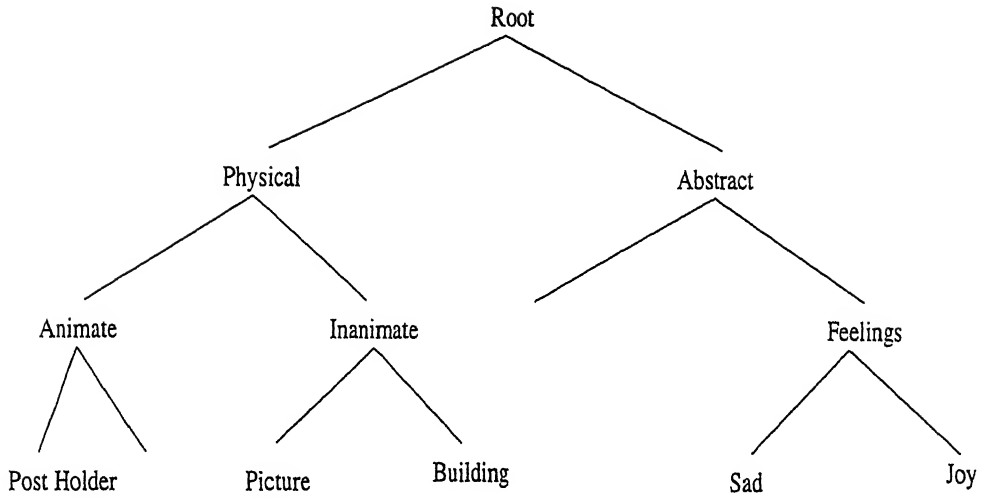


Figure 2.1: A Semantic Tree

is based on the type of verb part. Next level of partitioning is based on the number of syntactic units. All the examples are retrieved which are there in a sub-partition. First exact pattern matching is performed considering only syntactic information. Multiple examples may match in this case. Among these, most suitable one is picked up by calculating the distance. The example with the least distance is finally the one suited for translation. Distance d between IS (input sentence) and ES (example sentence) is calculated as -

$$d(IS, ES) = \sum_{p=1}^{n} d_p(ISG, ESG) + d_v(ISV, ESV)$$

where ISG stands for Input sentence noun syntactic group and ESG stands for example sentence noun syntactic group. Similarly, ISV and ESV stands for Input and example sentence verb groups. Other parameters are defined as below -

n = Number of noun syntactic group

$d_p(ISG, ESG)$ = distance between pth noun syntactic group of input and example source language sentence

$$= \frac{w1*ATD+w2*ASD}{w1+w2}$$

$$ATD = \frac{q1*SD+q2*(GD+ND+PD)}{q1+q2}$$

$$d_v(ISV, ESV) = \text{distance between verb group of input and example source language sentence}$$

$$= VCD$$

where $w1$, $w2$, $q1$ and $q2$ are weights assigned to sentence parameters. Weights can be adjusted to achieve the desired effect on translation. ATD, SD, GD, ND, PD, ASD and VCD are attribute difference, status difference, gender difference, number difference, person difference, additional semantic difference and verb category difference. Above method for calculation of distance has been taken from [Jai95].

2.2 Suitable Processor Architecture

String searching and pattern matching form the basis for many applications such as document processing, natural language processing, relational databases etc. Various hardware schemes have been proposed for these applications. These hardware differ in architecture but they all perform string searching and pattern matching in one way or other. Hardware schemes [MHT88] can be broadly classified into four categories - Content Addressable Memory (CAM) based, Cellular Array (CA) based, Finite State Automata (FSA) based and Dynamic Programming (DP) based. CA and DP based architectures have limitation of lesser flexibility and low comparison rate respectively. CAM and FSA based architectures offer practical advantages.

2.2.1 CAM Based

In this approach, input string is simultaneously matched with all the patterns stored in associative memory. Associative memory is loaded, subsequently, with new data to continue the comparison if needed. Since parallel comparisons are performed in this scheme, it has advantage of having faster comparison rate. But it doesn't offer

flexibility. Implementation of different search criteria, such as don't care conditions, fixed/variable length match, doesn't perform well in this approach.

2.2.2 FSA Based

In this approach, comparison is performed in steps and outcome of comparison triggers state transition in FSA whose state table may be stored in RAM or may be hard-coded in processor itself. This approach is suitable for implementing complicated search criteria but it suffers from low comparison rate. This approach offers performance advantage over CAM based approach for complicated search criteria but it performs poorly for simple search criteria such as exact string match.

2.3 Present State of Work

Most of the hardware schemes for string search and pattern matching are combination of CAM and FSA based approach. Ottmann et. al. [TAOS82] has proposed a design of a dictionary machine which is suitable for VLSI implementation. His machine supports SEARCH, INSERT, DELETE and EXTRACTMIN operation on an arbitrary ordered set. P. Sukumar et. al. [Suk96] have designed processors for dictionary search (named as DiSe) and example matching (named as MediCal). DiSe and MediCal are instructionless processor and, so, they offer limited flexibility. Burkowski et. al. [Bur82] has proposed a hardware using off the shelf chips for text retrieval document. His design involves hardware based hashing scheme. Yamada et. al [MHT88, HYT87] has proposed a scheme which is CAM and FSA based. Their scheme can perform complex matching such as approximate comparison, nonanchor search, variable length comparison and multiple variable length *don't care* search. Davis et. al. [DL86] has proposed a scheme for constructing the search algorithm for associative memories.

String search and pattern matching also find application in artificial intelligence such as natural language processing, problem solving and theorem proving. Few hardware schemes have been proposed for these applications also. Naganuma et. al.[JNK88] has given a CAM based architecture for Prolog machine. It uses CAM

based backtracking and achieves speed up through a hierarchical pipeline. David [SL91] has proposed a chip design which can be used as a coprocessor for tree pattern matching. It uses combination of content addressable memory, shift register and one bit wide stack.

Few processor architectures have been proposed which achieve stronger abstraction at hardware level. They deviate from classical von Neumann architecture. Lopriore [Lop84] proposed capability based tagged architecture which implements different level of abstraction. Three levels, i.e., lower, intermediate and upper. provide support to the machine type, pre-defined type and user defined type respectively. Hennessy has presented investigation of architectural design of VLSI processors in [Hen84].

2.4 Summary

The HEBMT approach offers advantage in terms of speed and disk space over example based approach. In this approach, translation is carried out by first analysing the input sentence and then matching it with abstracted examples. Performance of the HEBMT approach can be, further, improved by performing dictionary search and example matching in hardware. For this kind of application, most of the hardware schemes have been proposed around CAM based approach or FSA based approach due to their practical advantages. FSA based approach has been chosen in present work due to it's flexibility advantages.

Chapter 3

DESIGN ISSUES AND APPROACH

In machine translation application, dictionary search and example matching are basic operations. Speed of translation is limited as performing dictionary search and example matching take time. Translation can be sped up by having a parallel architecture and by having specific hardware for basic operations. With multiple processors, MT application can be performed in parallel, i.e., processors may translate a single senetence in parallel or they may translate multiple sentences in parallel. Parallel architecture will have specific hardware for performing basic operations to make them faster. MT software running on multiple processors can interact with specific hardware to perform dictionary search and example matching.

Present thesis proposes design of a specific hardware for dictionary search and example matching. Prior to hardware designing, issues related to MT application were examined and a parallel architecture was conceptualised. Some of the units, in parallel architecture, were performing basic operations. These units were selected for designing specific hardware. Conceptualisation of parallel architecture helped in determining services of specific hardware. In present chapter, issues, taken into consideration during hardware designing, have been examined first. And later, a parallel architecture has been presented. Finally, the chapter concludes by describing the approach taken to design the hardware.

3.1 Issues

As mentioned earlier, performance speed up, in translation, can be obtained by exploiting parallelism and by having specific hardware for dictionary search and example matching. Though simple search and matching criteria can be implemented at hardware level, complicated criteria can be made in software. Hardware design should be flexible so that complicated criteria are supported. Following sub-sections describe the issues related to basic operations, parallelism and flexibility.

3.1.1 Dictionary Search

Dictionary search is a major operation in MT applications. During morph analysis, words in input sentence are searched in the dictionary. If the word is found in the dictionary, it's characteristics is retrieved and it is passed as the result of search. Characteristics of word could be it's type, meaning in target language, semantic tag etc. Different type of words will have different type of characteristics. So, in the dictionary base, there will not only be words but also some information associated with them. For dictionary search, hardware will receive a word as input. Subsequently, it will fetch words from dictionary base. For each fetched word, it determines whether it satisfies the search criteria. If the criteria is satisfied, information associated with fetched word is passed as the result.

Normally, the search criteria for dictionary search is exact match. But performance enhancement, at functional level, can be sought by having complicated criteria. For example, to obtain root word of *Abhushano*, word is transformed into several words and each is searched in the dictionary. Instead, if search is carried out for first few letters with other letters being don't care, time to transform the word and to search multiple words can be saved. Other than the search operation, insertion and deletion of words are also performed on the dictionary base. For MT application, search operation is the most frequently used operation.

Two more properties of dictionary can be used to speed up search operation - 1. Partitioning the dictionary base to narrow down search and 2. Keeping the words sorted to perform binary search. Dictionary base can be partitioned at several level.

One possible partitioning scheme may use type of word at first level, length of word at second level and starting alphabet at third level.

3.1.2 Example Matching

Example matching is carried out in two steps. In first step, only syntactic units are matched and, in second step, distances are calculated for all examples which have same syntactic units but different attributes. Hashing can be used for the first step. Example base is partitioned where all examples, in a partition, have same syntactic units. A table, containing addresses of all partitions, is maintained. Hash function will take syntactic units of a sentence as input and will return an index into the table. In second step, distances are calculated for all examples in the partition, whose address has been obtained in the first step. One suggested distance calculation method has been given in the previous chapter which can be re-written as -

$$d(IS, ES) = \sum_{p=1}^{n} d_p(ISG, ESG) + d_v(ISV, ESV) \quad (3.1)$$

$$d_p(ISG, ESG) = W1 * ATD + W2 * ASD \quad (3.2)$$

$$ATD = Q1 * SD + Q2 * (GS + ND + PD) \quad (3.3)$$

$$d_v(ISV, ESV) = VCD \quad (3.4)$$

$$W1 = w1/(w1 + w2) \quad (3.5)$$

$$W2 = w2/(w1 + w2) \quad (3.6)$$

$$Q1 = q1/(q1 + q2) \quad (3.7)$$

$$Q2 = q2/(q1 + q2) \quad (3.8)$$

where each term have there usual meaning. ASD, SD, GS, ND and PD are attribute differences which can take only few values. For example, SD can take only 0 or .5. Multiplication of attribute difference with $W1$, $W2$, $Q1$ or $Q2$ would leave their values limited in number. It simplifies equation (3.2) and (3.3) as -

$$d_p(ISG, ESG) = ATD' + ASD'$$

$$ATD' = SD' + GS' + ND' + PD'$$

where each term i.e. SD', GS', ND', PD' and ASD' will take few values which can be precomputed and put into the memory. This way, distance calculation can be done using addition only. Removal of division and multiplication operation in distance calculation eases their implementation in hardware. Partitioning can also be applied to the example base. Possible criteria for partitioning could be number of syntactic units, syntactic unit types etc.

3.1.3 Flexibility

MT applications will use services of hardware for dictionary search and example matching. Different MT applications require different ways of searching and matching. If hardware is tied to one scheme of searching and matching, it may be impossible or difficult to implement different MT applications on top of it. Future work in area of machine translation may result in different search and match criteria to perform translation efficiently. Hardware design should be flexible enough so that it can support different MT applications.

3.1.4 Parallelism

Performance enhancement, in translation, can be brought about at three levels -

- By having specific hardware for dictionary search and example matching
- By having multiple units for dictionary search and example matching to execute multiple search and match requests in parallel.
- By having multiple processors to execute MT software in parallel. For example, morph analysis can be performed in parallel by breaking the sentence or multiple sentences can be translated in parallel.

Hardware design, for basic operations, should be able to fit in a parallel architecture.

Size	Hit Ratio
50	43.4
100	56.1
150	62.2
200	65.8
250	69.0
300	70.1

Table 3.1: Cache Hit Ratio

3.1.5 Other Design Issues

Few architectural issues were considered for designing the processor. System will largely be used for translating the document in which sentences are related. Many words are repeatedly used in the document. Cache can be used to hold words which are repeatedly used. Simulation was carried out to determine the cache size and result indicated that cache size of 256 words will give about 69% hit ratio. Table 2.1 shows hit ratio obtained for different cache size. 32-bit processor architecture has been chosen for designing.

3.2 System Architecture

Processor design was preceded by designing of a parallel architecture (Fig 3.1). System is divided into two parts - Memory Unit and Translation Unit. Memory unit provides support to the basic operations, i.e., dictionary search and example matching. It accepts requests for searching the words in the dictionary or for matching the pattern of a sentence with that in example base. Translation unit uses the services of memory unit to carry out translation. While parsing a sentence, translation unit generates requests for searching words in the the dictionary. Requests are passed to memory unit which performs the search operation and passes the result back to translation unit. Once analysis is over and syntactic units have been identified, input sentence pattern is once again passed to the memory unit which matches it with examples in example base and passes the result of the matching back to the

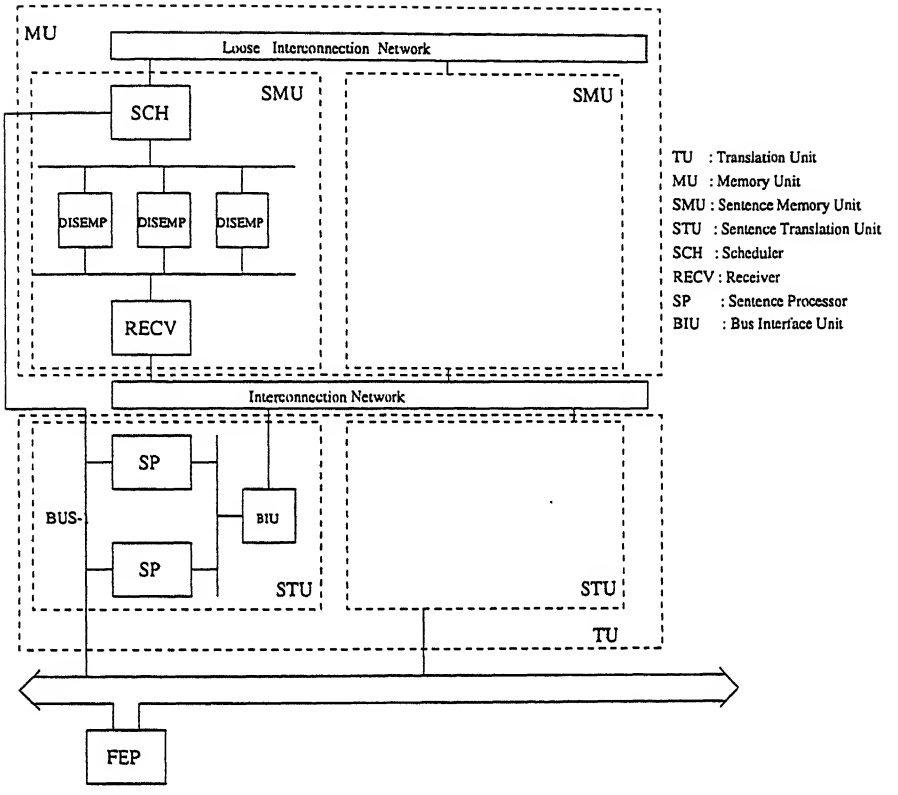


Figure 3.1: The System Architecture

translation unit.

Translation unit consists of sub-units known as sentence translation unit (STU) and, similarly, memory unit consists of sentence memory unit (SMU). Every STU has a SMU associated with it which accepts and processes it's request. Sentences to be translated are distributed among STU's by front end processor (FEP). Multiple STU and SMU exploit parallelism at sentence level. To perform translation of a sentence in parallel, STU consists of multiple processors (sentence processors or SP). SPs will be executing software for MA, ISG and EM phase. Each of the phase can be performed on multiple SPs and they may coordinate among themselves through BUS-1. SMU consists of multiple specific processors (DIctionary Search and Example Matching Processor or DISEMP). DISEMP searches a word in the dictionary and matches example in the example base. Request to SMU is received

by scheduler (SCH) which distributes it among DISEMPs. For example, if following sentence is to be translated -

vaha ja raha he

SP of the first STU will be receiving the request from FEP. The SP carries out morph analysis and generates requests for search of word *vaha* and *jana* in the dictionary. Requests go to SCH1 which may distribute search of word *vaha* to DISEMP1 and search of word *jana* to DISEMP2. Result of a search request or a match request from DISEMP is received by receiver (RECV) which passes it forward to SP. On receiving the characteristics of words, SPs identify syntactic units, and pattern of above sentence is determined as -

$$np1[S, M, A]\, mv[T]$$

Above pattern is once again passed to the memory unit and it goes through the same cycle for example matching and finally SP get the target language pattern based on which translation is carried out and given to FEP.

SCHs are loosely coupled to pass on the request to another SCH for load balancing. During morph analysis, a word is transformed into multiple words and all of them are searched as a candidate root word. If any of them is found, others need not be searched in the dictionary. To fecilitate this, SCH receives information from DISEMPs and, based on it, it may or may not pass further requests to DISEMPs.

Functionality of the memory unit is divided between SCH and DISEMPs. As it has been mentioned that dictionary base and example base are partitioned, partitioned database will be residing with DISEMPs. A partition has certain properties such as all words may start with a particular alphabet or they may have same length. Properties of partitions are known to SCH. For example, if SCH receives a request for searching the word *ghara*, it may ask DISEMP to search it in the partition in which all words are of length five. Similarly, in case of example matching, DISEMP will only perform distance calculation while SCH will perform syntactic unit match.

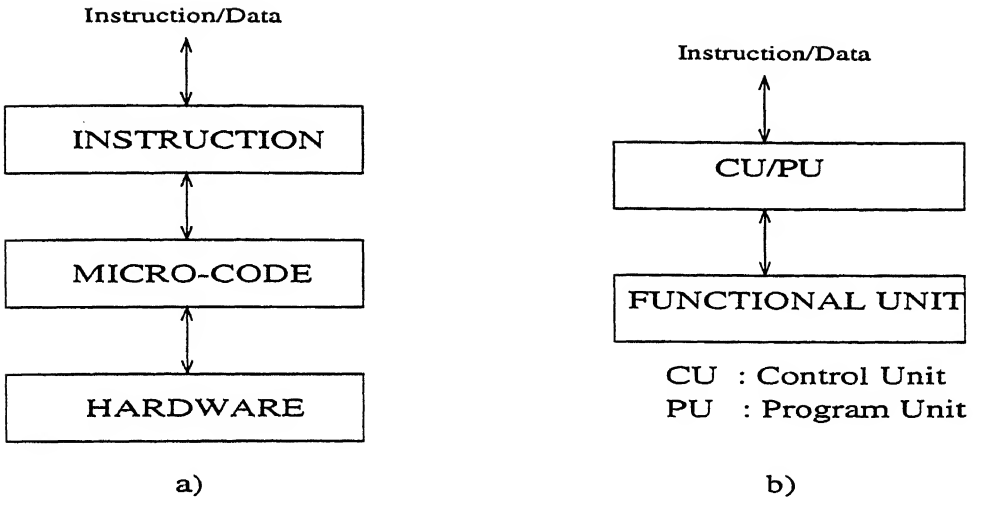


Figure 3.2: Three Levels in Functionality

3.3 Approach to Processor Design

3.3.1 Flexibility

As mentioned in previous chapter, two primary approaches have been suggested for string searching and pattern matching - CAM based and FSA based. FSA based approach offers advantage in terms of flexibility. Since flexibility is a major issue in our design, FSA based approach has been chosen. To take advantage of the fact that words in the dictionary are sorted, FSA based scheme also makes it possible to implement binary search.

Flexibility comes at the cost of performance. In conventional architecture, functionality is provided at three levels (Fig 3.2) - hardware, micro-code and instruction. The hardware level is fastest but it doesn't give flexibility. Flexibility is incorporated either through micro-code or through instructions. At higher levels such as instruction level, flexibility increases at the cost of performance. At instruction level, performance degradation occurs due to overheads such as instruction fetch, instruction decode etc.

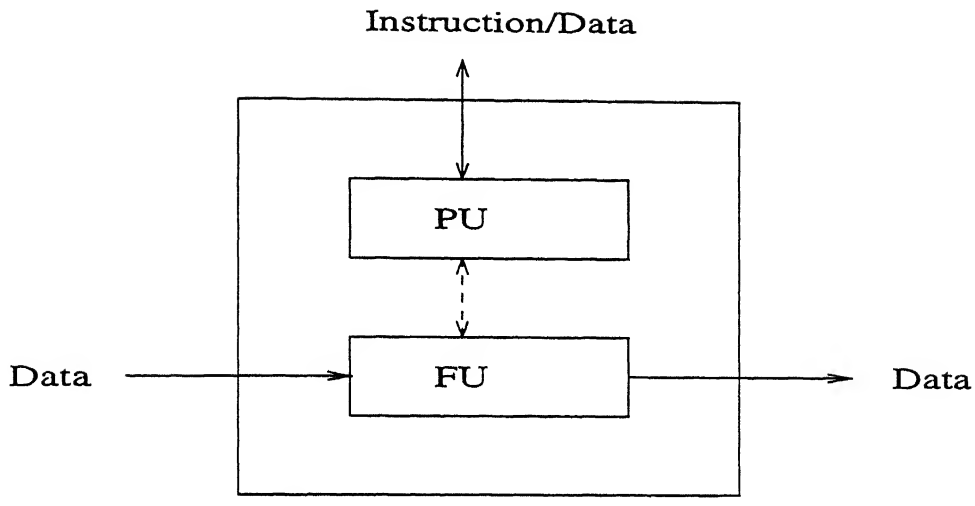


Figure 3.3: Modified Architecture

3.3.2 Approach to Incorporate Flexibility

Let F denote set of functionalities required in processor. Some of the functionalities in F are used frequently and their implementation have to be faster in processor. For small F, hardware level is sufficient, and processor need not support instructions. As size of F increases, higher levels of implementations have to be chosen. Limitation of the architecture shown in Fig 3.2, is that due to support to instructions, most frequently used functionalities suffer from it's overhead.

Most frequently used operations, in hardware, can be performed in lesser time by passing data to hardware directly, instead of through instructions. Modified architecture is shown in Fig 3.3. Here, most frequently used operations are implemented in hardware, and they directly receive data from external to processor. Processed data is directly passed to the external environment. Let's assume that hardware implements the function f on input data x and $y = f(x)$. Function f has been implemented as three sub-functions $f1$, $f2$ and $f3$ which operate on x in that order. Intermediate values between sub-functions are held in registers within functional unit.

In this approach, program unit exercises lesser degree of control over functional units (or hardware). Program unit and functional unit are loosely coupled. Program

unit can interact using following with functional units -

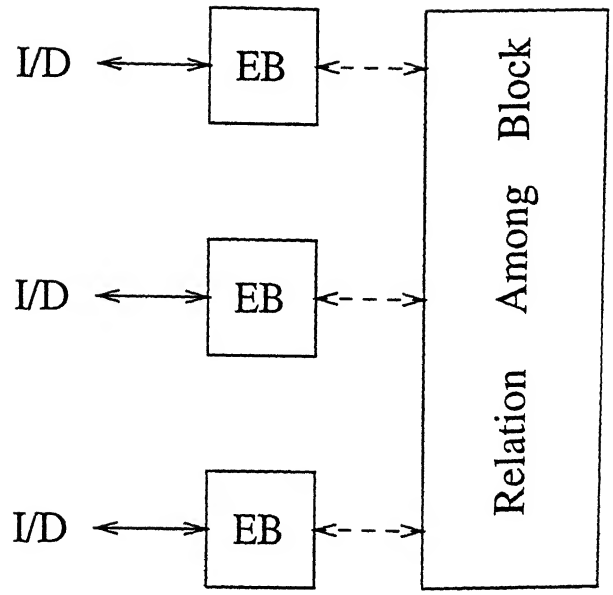
- Micro-Instructions
- Interrupts

Program unit can change the sequence of sub-functions using micro-instructions, i.e., $f1$, $f2$ and $f3$ can be executed in different order depending on the micro-instruction. The micro-Instruction register specifies the sequence of sub-functions. Program unit can change intermediate values between subfunctions using interrupts. After performing a sub-function, functional unit stops and interrupts program unit. Program unit can examine registers and can modify it appropriately. It further signals the functional unit to continue the execution. For example, if functional unit stopped after sub-function $f1$ and subsequently program unit applied function g to intermediate value, result y will be obtained as $f1$,g,$f2$,$f3$. Program unit has it's own execution units and registers to process intermediate values. Advantage of this approach is that functional unit can be made to operate without interference from program unit for most frequently used operations. In this case, instruction overheads will not be there. Least frequently used functionality is implemented in software through program unit.

3.3.3 Extension of Idea

As discussed earlier, the processor architecture consists of two blocks - program unit and functional unit. Program unit has it's control unit, registers and execution unit. It fetches instruction from memory and processes data accordingly. Functional unit also has it's own control unit, registers and execution unit. It's control unit is not as complicated as that of program unit. And it only receives data from external to processor. Program unit and functional unit interact through micro-instruction and interrupts.

Above concept can be generalised and processor architecture can be viewed as consisting of multiple blocks. Each block will have it's own control unit, registers and execution units. Each block will be receiving instruction, data or both from external



I/D : Instruction/Data

EB : Execution Block

Figure 3.4: A Generalised Processor Architecture

world. Multiple blocks within processor, are related through micro-instruction and interrupts (Fig 3.4).

3.4 Summary

In this chapter, issues related to hardware design were examined. Implementation of dictionary search and example matching should be flexible to support different criteria. Partitioning can be used to narrow down search space and matching space. Hardware design is to fit in a parallel architecture. A parallel architecture has been presented to determine the services of hardware.

Flexibility has been chosen as an important consideration for designing. Flexibility, in the hardware, is incorporated by providing instructions. In a flexible design, performance degrades due to instruction overheads. Overheads can be minimised

for frequently used operations by distributing functionalities among multiple blocks. Multiple blocks are loosely coupled through micro-instructions and interrupts.

Chapter 4

EXTERNAL ENVIRONMENT TO DISEMP

DISEMP is a DIctionary Search and Example Matching Processor. It can be used in a parallel environment. In previous chapter, a parallel architecture was discussed which uses DISEMP. DISEMP receives request from scheduler, processes it and passes the result to receiver. Database consisting of dictionary and examples reside in memory. Request packet, result packet and memory form external environment to processor. In present chapter, their formats have been described.

4.1 Overview of DISEMP

It has three external buses (Fig 4.1) - EREQ Bus (External Request Input Bus), ERES Bus (External Result Bus) and EM Bus (External Memory Bus). EREQ and ERES bus are 8-bit bus while EM bus is 32 bit bus. DISEMP receives request from scheduler through EREQ Bus. Request is for searching a word or for matching an example. Dictionary base and example base reside in memory. Data from memory is fetched through EMbus. Result of a request is transmitted to receiver through ERES bus. For example, if the request is for searching a word, DISEMP fetches words from memory through EM bus. If search criterion is successful for fetched word, information associated with it, is transmitted on ERES bus. Both binary

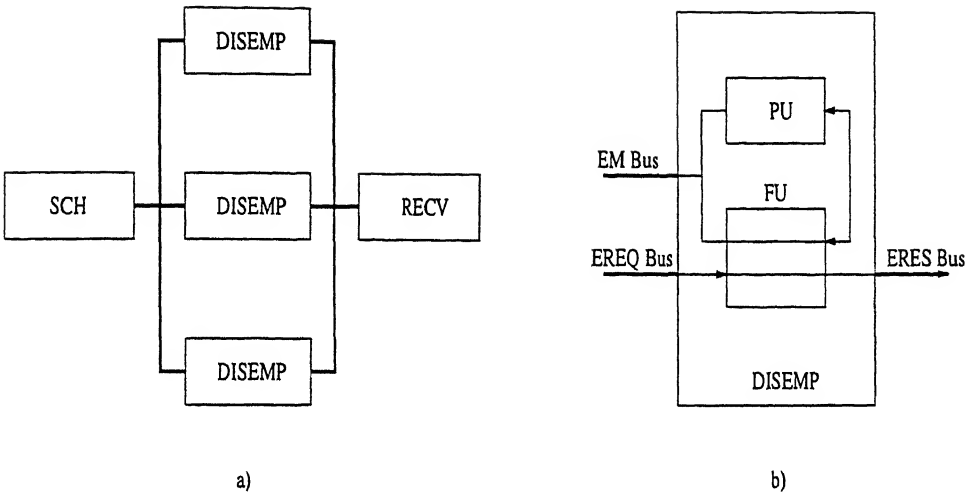


Figure 4.1: External Interface

and sequential search can be performed for a word request. Similarly, for example matching operation, it calculates distances of examples from input pattern. The example, with least distance, is chosen and it's information is passed as result.

Internally, DISEMP has two units for searching the word and one unit for matching the example. With each unit, two buffers are associated where arrived requests reside. It has a cache of size 256 words which makes use of repetition of words in a document. DISEMP supports partitioning of dictionary base and example base. It is a 32 bit architecture with address space of 16 MWords. DISEMP has a program unit which executes instruction read from memory. Program unit interacts with search and match unit to perform complicated operations such as substring matching.

4.2 Interface with External Environment

As mentioned earlier, request packet, result packet and memory form external environment of DISEMP. In this section, their formats have been described.

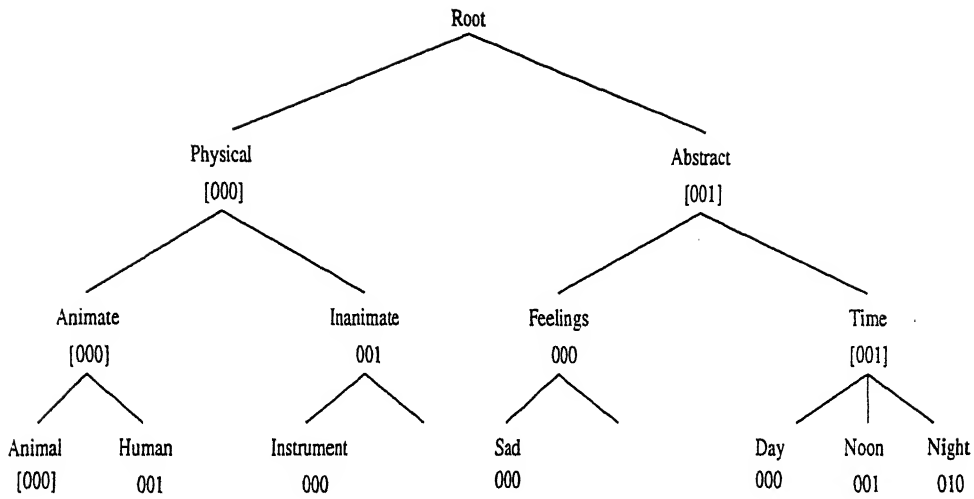


Figure 4.2: Semantic Tree

4.2.1 Request and Result Format

DISEMP accepts a word for searching it into the dictionary (i.e. a search request) or a sentence pattern for matching it with examples (i.e. a match request). Requests are further classified into two categories - Predefined and User defined. A predefined request is processed by functional unit (i.e. word search unit or example matching unit) alone. While user defined requests are interpreted by program unit. Program unit interacts with functional unit to process a user defined request. User defined requests are used to perform complicated and least frequently used operations which require PU involvement. For example, to determine all words with a given length, user defined request is used. Request size is fixed at 20 bytes or 5 words. First word of a request contains header and next four contain a dictionary word or a sentence pattern. To process a request, funtional units fetch words or examples from database. Whichever word or example satisfies criteria, is selected and information associated with them is passed as result. Request header is also transmitted as part of the result. Size of the result packet is not fixed but it can not be more than 256 bytes.

Following is the format of request header for a search request -

Bit[0- 0] : 0-Predefined ; 1-User Defined Request

Bit[1-15] : Request identification Number

Bit[16-23] : Partition Number in which search will be made
Bit[29-29] : 0-Sequential Search ; 1-Binary Search

Following is the format of request header for a match request -

Bit[0- 0] : 0-Predefined ; 1-User Defined Request
Bit[1-15] : Request identification Number
Bit[16-23] : Partition Number in which matching will be done

In a match request, sentence pattern contains attributes and semantic tags of syntactic units. In a pattern, three bits are allocated for each unit of information and each word contains 10 such units. An attribute such as gender, number etc. is represented in 3-bit. While a semantic tag occupies multiples of 3-bit depending on the height of semantic tree. For example, let first word of an example be

000 001 001 010 000 001 000 000 010 001 00

In this example. 000 in first three bits may indicate MASCULINE if first attribute type is interpreted as GENDER. Semantic tags are also interpreted in units of three. If height of semantic tree is 3, a semantic tag occupies three contiguous 3-bit units where each unit indicates what is the position at each level. In above example, if second, third and fourth units are interpreted as semantic tags, 001 001 010 indicates semantic tag *night* for the semantic tree shown in Fig 4.2.

4.2.2 Memory Map

Memory contains the dictionary base, example base and programs (Fig 4.3). Dictionary base and example base are partitioned and information about each partition is there in partition table. Partition table contains 256 entries. It starts from address 000000(Hex) and each entry in partition occupies 2 words. An entry has following format -

First Word Bit[23- 0] : Starting Address of Partition
Second Word Bit[23- 0] : Number of Entries
Second Word Bit[31-24] : Length of record (in words)

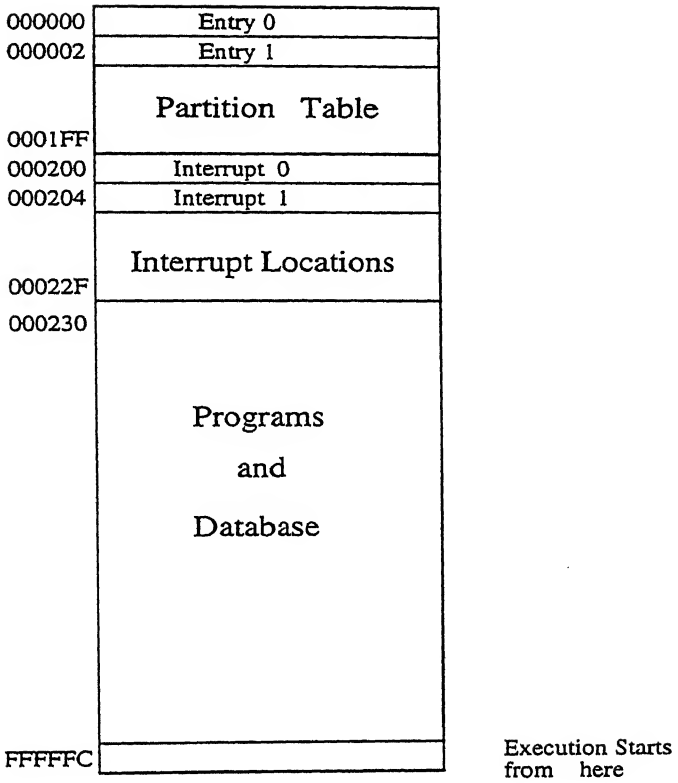


Figure 4.3: Memory Map

DISEMP always starts the execution from address FFFFFC(Hex). Some of the programs in memory reside in form of interrupt service routine. Interrupts, in DISEMP, are used for interaction between program unit and functional unit. Functional units suspend it's operation and interrupt program unit so that it can modify it's registers. Interrupt service routines are used to implement different types of search and match operations. Functional units can interrupt program unit under four conditions -

1. When request arrives
2. When an example or word has been fetched
3. When comparison or matching has been performed
4. When processing of request is over

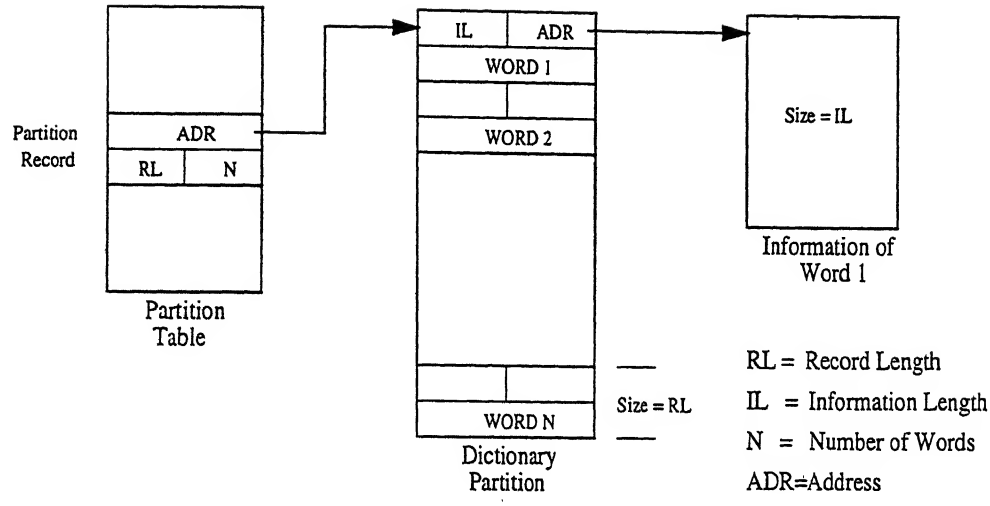


Figure 4.4: Structure of Dictionary Partition

With three functional units, there will be 12 interrupt types. For each interrupt, four words in memory are reserved which contain jump instruction to interrupt service routine. When program unit is interrupted, it saves program counter into stack and jumps to location reserved for interrupt. Fig 4.3 shows memory locations for interrupts.

4.2.3 Database Format

Dictionary base and example base form the database. Dictionary base consists of words and information associated with them. Structure of dictionary partition is shown in Fig 4.4. Each entry in a dictionary partition has following format -

First Word Bit[23- 0] : Address of information associated with this word
First Word Bit[31-24] : Length of information in words
Following Words : Actual Word

An example partition contains weights, attribute types and examples. Fig 4.5 shows structure of an example partition. An example partition starts from the address whose lower four bits are zero. Maximum of 32 weights can be there where each word stores 4 weights. So weights can take values from 0 to 255. After weights,

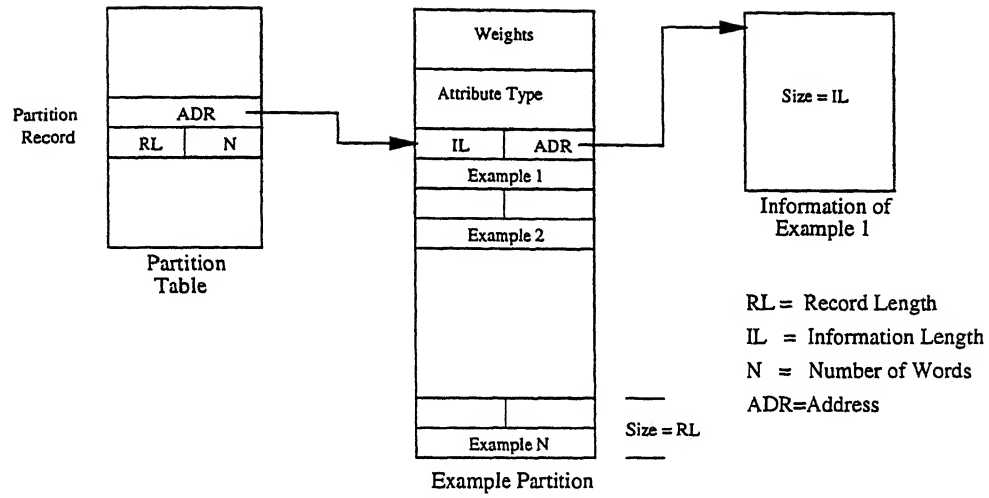


Figure 4.5: Structure of Example Partition

attribute types are stored. Each attribute type takes single byte whose format is as shown below -

Bit[7-6]	:	00-Use four bits from this byte for weight index
		01-Use three bits from this byte for weight index
Bit[5-2]	:	Used to determine weights if Bit[7-6]=00
Bit[4-2]	:	Used to determine weights if Bit[7-6]=01
bit[1-1]	:	0-Attribute is not semantic tag
		1-Attribute is semantic tag

An example record, in an example partition, is stored similar to word partition. First word of record contains address and length of information associated with the example. Rest of the words contain example pattern. Format of an example record is -

First Word Bit[23- 0]	: Address of information associated with this example
First Word Bit[31-24]	: Length of information in words
Following Words	: Actual Example

An example pattern contains attribute and semantic information. Attributes and semantic tags are properties of syntactic units. An attribute, in an example

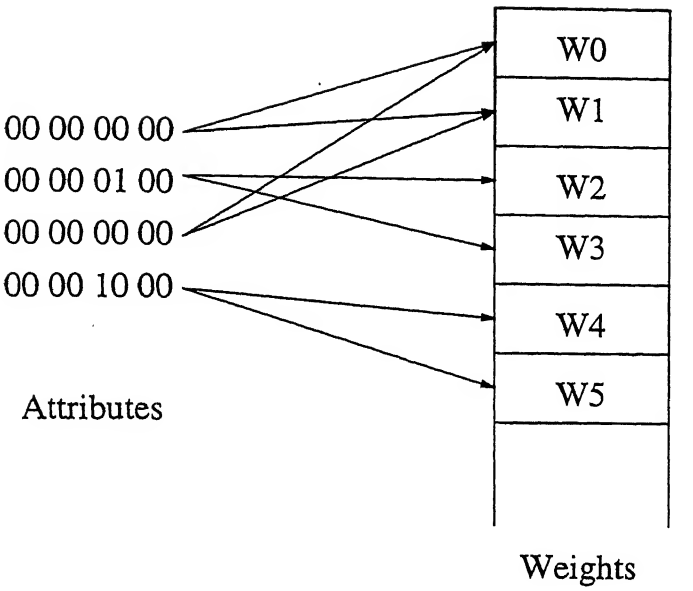


Figure 4.6: Sample Attributes and associated weights

pattern, occupies 3-bit but a semantic tag occupies multiples of 3-bit. As mentioned earlier, an example partition consists of weights, attribute types and example record. Attrubutes and semantic tags, in an example pattern, are interpreted using attribute type. For example, Fig 4.6 shows few attribute types and weights in binary format. Fig 4.6 also shows weights associated with each attribute type. Let following be an example of input pattern and example pattern -

Input pattern : 001 001 000 001 010 000 000 001 000 000 00
Example Pattern : 010 001 001 001 001 000 001 001 000 001 00

Above pattern is interpreted with attribute types shown in Fig 4.6. If first attribute type, i.e., 00, represents GENDER, 001 may denote MASCULINE and 010 may denote FEMININE in above patterns. First attributes of input and example sentence don't match i.e. outcome of matching will be 1. So, weight at position 00001, i.e., $W1$ is used in distance calculation. Final distance is obtained by summing the weights for all attributes. Exact algorithm for the attribute matching is given Fig 4.7.

Output :

Weight After Attribute Matching

Input :

1. *Input Sentence Attribute* s_{2-0}
2. *Example Sentence Attribute* e_{2-0}
3. *Attribute Type* a_{7-0}
4. *Ordered Set of Weights* $\{w_i \mid 0 \leq i \leq 3\}$

Notation :

$V[x_{7-0}]$ *: Value represented by bit pattern* x_{7-0}

Algorithm :

Case A. a_{7-6}*=00*

1. *if(*s_{2-0}*=*e_{2-0}*) pos=V[*a_{5-2}*0]*
2. *if(*$s_{2-0} \neq e_{2-0}$*) pos=V[*a_{5-2}*1]*
3. *Output* w_{pos}

Case B. a_{7-6}*=01*

1. *if((*s_{2-0}*-*e_{2-0}*)=0)*
   pos=V[a_{4-2}*00]*
2. *if((*s_{2-0}*-*e_{2-0}*)=1 or (*e_{2-0}*-*s_{2-0}*)=1)*
   pos=V[a_{4-2}*01]*
3. *if((*s_{2-0}*-*e_{2-0}*)=2 or (*e_{2-0}*-*s_{2-0}*)=2)*
   pos=V[a_{4-2}*10]*
4. *if((*s_{2-0}*-*e_{2-0}*)=3 or (*e_{2-0}*-*s_{2-0}*)=3)*
   pos=V[a_{4-2}*11]*
5. *Output* w_{pos}

Figure 4.7: Algorithm for an Attribute Matching

4.3 Summary

DISEMP is externally interfaced through three external buses. Two buses are used for receiving the request and for transmitting the result. One bus provides an interface with memory. Memory contains the dictionary base and example base. Present chapter has discussed format of external environment i.e. request packet, result packet and memory. Internal details of DISEMP are described in following chapters.

Chapter 5

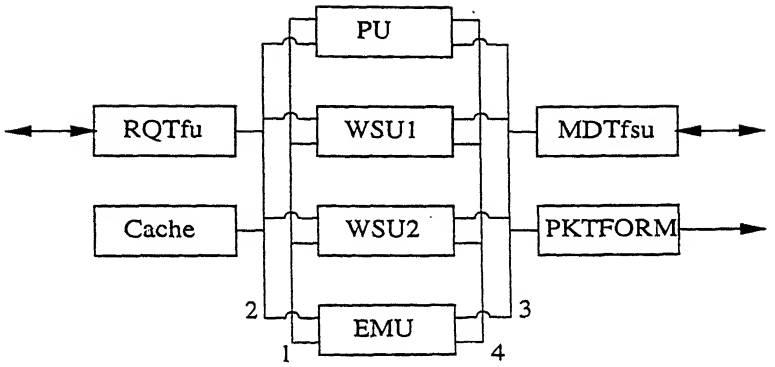
DISEMP DESIGN

Present chapter discusses the design of DISEMP. DISEMP has been designed with flexibility consideration so that it can handle complicated search and match operations such as substring matching. The design incorporates some performance enhancement features such as partitioning and caching. In this chapter, first the architecture of DISEMP is described. Subsequently, we describe the registers and their contents. Finally, the chapter concludes with the description of instruction set.

5.1 DISEMP Architecture

5.1.1 Overall Architecture

Overall architecture of DISEMP is shown in Fig 5.1. It has two Word Search Units (WSU) for searching the words, i.e., WSU1 and WSU2. There is a single unit for example matching, viz, EMU (Example Matching Unit). DISEMP has three external buses - EREQ Bus, ERES Bus and EM Bus. Interface to EREQ Bus is provided through RQTfu (Request Fetch Unit). Request, coming on the EREQ bus, is received by RQTfu, and is further transmitted to the functional units (i.e. WSU1, WSU2 or EMU). Functional unit processes the request and passes the result to PKTFORM Unit. PKTFORM Unit transmits the result on ERES Bus. Similarly, EM Bus is interfaced through the MDTfsu (Main Data Fetch Store Unit). EM bus connects the DISEMP and external memory. The external memory contains



1. PC Bus
2. RI Bus
3. DF Bus
4. INTR Bus

Figure 5.1: Overall Architecture

dictionary base, example base and programs. MDTfsu fetches words from dictionary base and passes it to the WSU1 or WSU2. It fetches examples and passes it to the EMU.

DISEMP is programmable. It has a program unit (PU) which receives the instructions from memory and executes it. The program unit interacts with the functional units (i.e. WSU1, WSU2 and EMU) to perform operations which can not be handled by functional units alone. Registers, in DISEMP, are distributed among PU and these functional units (WSU1, WSU2 and EMU). Word Search Unit has registers for storing the search requests and for storing the word fetched from memory. Registers, in EMU, store match requests, example and weights (used in distance calculation). PU has registers which can be used by software. On arrival of the request to search a word, it is first looked into the cache. If the word is not found in the cache but is found in the dictionary, it is written into the cache.

Various units, in DISEMP, are connected through four internal buses - PC Bus (Program Controlled Bus), DF Bus (Data Fetch Bus), RI Bus (Request Input Bus) and INTR Bus (Interrupt Bus). PC Bus is used by the program unit to access the registers of WSU1, WSU2 and EMU. DF Bus is used for passing the data retrieved from memory to PU, WSU1, WSU2 and EMU. For example, to retrieve an example,

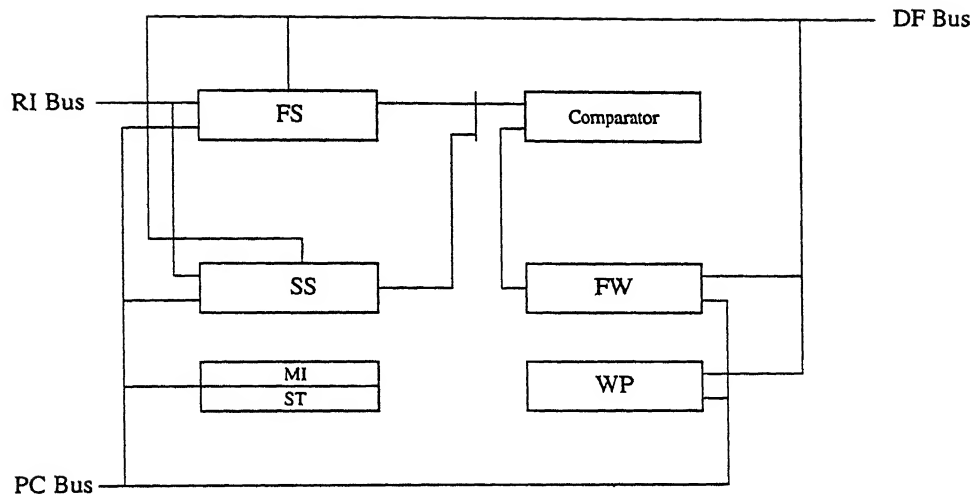


Figure 5.2: Architecture of WSU

EMU places the request with MDTfsu, and waits till MDTfsu finishes processing of existing request. Subsequently, EMU passes the address, from where example is to be fetched, to MDTfsu. MDTfsu retrieves the example and delivers it to the EMU through DF Bus. Similarly, RI Bus is used for transferring the request to functional units. RI Bus is also used by WSU1 and WSU2 to access the cache. INTR bus carries the interrupt signals which are used for interaction between program unit and functional units. Functional units interrupt PU, in between operation, so that PU can examine and modify it's registers. Interrupt occurs only if it is requested by PU.

5.1.2 Word Search Unit (WSU)

Word Search Unit (WSU) searches a word in a dictionary partition. Search can be made sequentially or binary. The architecture of WSU is shown in Fig 5.2. FS and SS contain the search requests. A search request contains a word and the partition number in which search is to be carried out. On the arrival of request, WP register is loaded with partition's information. Subsequently, the first word in the partition is fetched and kept in FW registers. Input word (residing in FS or SS) and fetched word are compared by comparator. If comparison is successful, word search unit requests

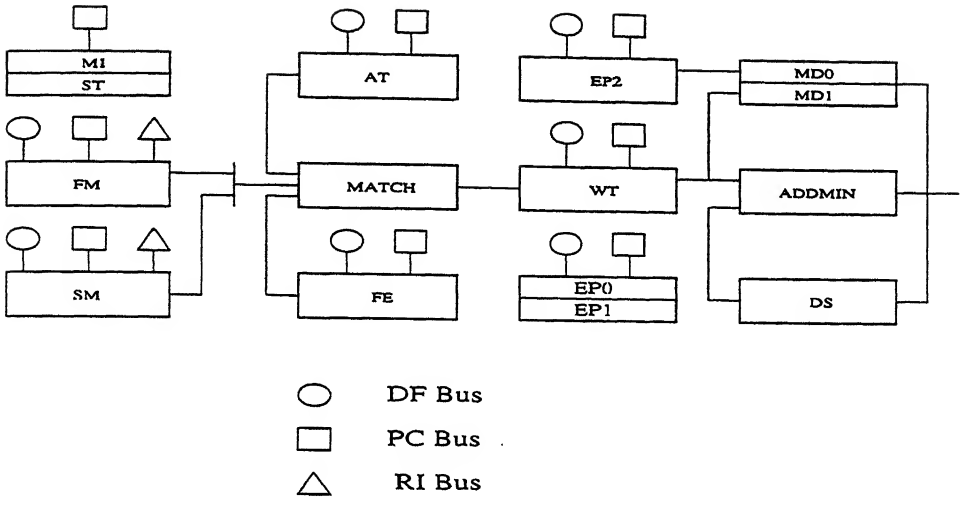


Figure 5.3: Architecture of EMU

MDTfsu to retrieve the word's information and pass it to the PKTFORM unit. Comparator can perform less than and equal operations. MI (Micro-Instruction) register is appropriately loaded to control comparison. ST register contains the status of Word Search Unit such as if buffers are holding requests, WSU is operating on which buffer, which operation will be performed next by WSU etc.

5.1.3 Example Matching Unit (EMU)

Example matching unit chooses the example which has least distance. Information associated with this example is passed as the result of matching. The architecture of EMU is shown in Fig 5.3. Match request resides in FM or SM registers. On the arrival of request, partition information is loaded in the EP registers. Subsequently, from the example partition, WT and AT registers are loaded with weights and attribute types respectively. Example pattern is fetched and brought into the FE registers. Unit MATCH matches an attribute of input sentence and example sentence. Output of MATCH, alongwith attribute type from AT register, provides an index into the WT register. Weight selected is added to the distance, and new distance is put in the DS register. Distance addition is done by ADDMIN unit. After distance has been calculated, it is compared with

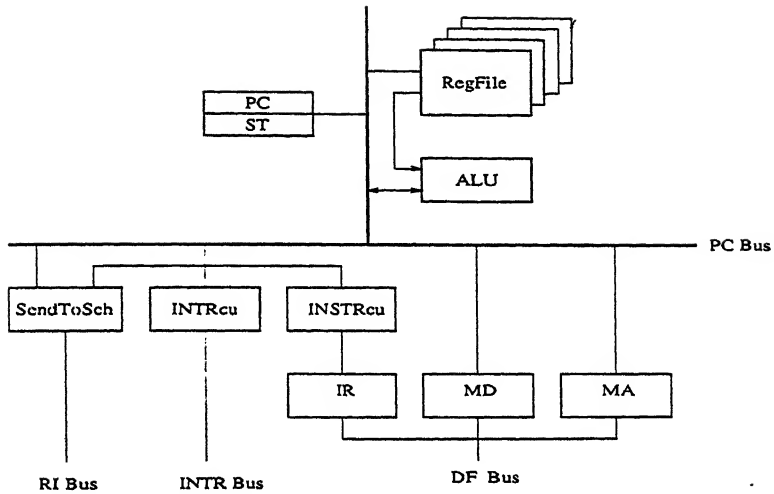


Figure 5.4: Architecture of PU

the minimum distance which resides in MD0 register. If new distance is smaller, MD0 is updated. The address of target pattern for this example is moved to the MD1 register. At the end, the target pattern, whose address is in MD1 register, is transmitted as the match result. MI register controls the matching operation. ST register holds the status of EMU.

5.1.4 Program Unit (PU)

The program unit receives instructions from memory and executes them. It has it's own control unit, registers and execution unit. The architecture of PU is shown in Fig 5.4. Instruction fetched, from memory, is transferred to the IR and MD registers. MA register supplies the address to MDTfsu in a memory access. INSTRcu decodes the instruction and generates appropriate control signal for it's execution. Data transfer within PU takes place through PC Bus. PC bus is also used for accessing functional unit registers. INTRcu receives interrupt from functional units and interrupts INSTRcu. On receiving an interrupt, INSTRcu makes a call to the interrupt service routine. ST register contains the flags such as CARRY, ZERO etc. It, also, contains the interrupt mask information. If any interrupt is masked in this register, it will not be generated by functional units. RegFile contains few

general purpose and few special registers such as stack pointer. SendToSch unit is used for transferring the information to scheduler.

5.2 Register Content

Register sets in DISEMP are distributed among WSU, EMU and PU. Each register is of 32-bit size. Following sub-section describes the registers in functional units and PU.

5.2.1 Registers of WSU

WSU registers are classified into six sets -

FS (First Search Request) Register Set
SS (Second Search Request) Register Set
FW (Fetched Word) Register Set
WP (Word Partition) Register set
MI (Micro-Instruction) Register Set
ST (Status) Register Set

FS and SS register sets contain the search request. There are ten registers in this set numbered from FS0 to FS4 and SS0 to SS4. FS0 and SS0 contain the request header. Other eight registers FS1 to FS4 and SS1 to SS4, contain the word to be searched. FW register contains the word which has been brought from the dictionary partition. There are four registers, numbered from FW0 to FW3, in this set.

WP set contains the partition information. There are three registers in WP set, numbered as WP0 to WP2. Contents of registers are as given below -

WP0(23-0) : Starting address of partition
WP1(23-0) : Number of records in partition
WP1(31-24): Record Length
WP2(31-24): Length of information associated with word
WP2(23-0) : Address where information about word is kept

MI register contains the micro-instructions to control comparison operation. It is divided into four parts of 6 bits each. Each group of 6-bits describes the operation to be performed by WSU in a step. Description of first 6-bits are as given below -

MI(1-0) : Register to be selected from FS or SS
MI(3-2) : Operation to be performed by comparator
MI(5-4) : Register to be selected from FW

ST register contains the status of WSU. It contains following information -

ST(2-0) : Modified by PU to denote next operation
ST(5-4) : Status of FS register set
ST(7-6) : Status of SS register set
ST(8-8) : Result of Comparison
ST(10-9) : Comparator operating on which set,i.e., FS or SS
ST(11-11): If unit has interrupted PU and waiting
ST(12-12): If unit is idle
ST(18-15): Operation being performed by WSU

ST(2-0) can be modified by PU only when WSU has suspended it's operation after interrupting PU. PU can specify one of the following value for it -

1. Let default action takes place
2. Fetch Word from dictionary partition
3. Execute Micro-instruction
4. Transmit Result
5. Stop processing of current request

Status of FS or SS register set could be one of the following -

1. Free, not containing any request
2. Locked by Program Unit

3. Locked by RQDTfsu

4. Containing Request

5.2.2 Registers of EMU

Registers of EMU are similar to that of WSU. These registers have been classified into following categories -

FM (First Match Request) Register Set
SM (Second Match Request) Register Set
FE (Fetched Example) Register Set
EP (Example Partition) Register set
MI (Micro-Instruction) Register Set
ST (Status) Register Set
WT (Weight) Register Set
AT (Attribute) Register Set
MD (Minimum Distance Example) Set
DS (Distance) Register

FM and SM register sets contain match request. There are ten registers in this set numbered from FM0 to FM4 and SM0 to SM4. FM0 and SM0 contain the request header. Other eight registers FM1 to FM4 and SM1 to SM4, contain the sentence pattern. FE register set contain the example, fetched from example partition. There are four registers in this set, numbered from FE0 to FE3.

EP Register set has same purpose as that of WP register set. Following information is contained in this -

EP0(0-23) : Starting address of partition
EP1(0-23) : End address of partition
EP1(24-31) : Record Length
EP2(24-31) : Length of information associated with example
EP2(0-23) : Address where information about example is kept

There are two Micro-Instruction registers. Each register holds 20 bits of information. Each bit indicates if any attribute is to be skipped in distance calculation.

EMU has a status register ST whose contents are described below -

ST(2-0) : Modified by PU to denote next operation
ST(5-4) : Status of FM register set
ST(7-6) : Status of SM register set
ST(8-8) : Result of Comparison
ST(10-9) : MATCH unit operating on which set, i.e. FM or SM
ST(11-11) : If unit has interrupted PU and waiting
ST(12-12) : If unit is idle
ST(18-15) : Operation being performed by EMU

ST(2-0) can be modified by PU only when EMU has suspended it's operation after interrupting PU. PU can specify one of the following value for it -

1. Let default action take place
2. Fetch example from example partition
3. Execute Micro-instruction
4. Transmit Result
5. Stop processing of current request

Status of FM and SM registers is similar to that of FS and SS registers of WSU.

WT registers contain weights. There are eight such registers of 32 bit each numbered from WT0 to WT7. AT registers contain attribute types. There are ten such registers numbered from AT0 to AT9.

MD is minimum distance example register set. There are two registers in this set numbered from MD0 to MD1. Their contents are as given below -

MD0(0-7) : Minimum distance
MD1(24-31) : Information length of example with minimum distance
MD1(0-23) : Address of information associated with example of minimum distance

DS register contains the calculated distance for the example in FE registers.

5.2.3 Registers of PU

PU has few special and 44 general purpose registers. Following are special purpose registers -

PC	: Program Counter
SP	: Stack Pointer
ST	: Status register. Contains flags and interrupt masks
R0	: Contents of this register is fixed at zero
R1	: Contents of this register is fixed at one

PC is a 24-bit register and other registers are 32-bit. ST register contains the flags and interrupt masks. It's format is as shown below -

ST(0-0)	: CARRY Flag
ST(1-1)	: ZERO Flag
ST(2-2)	: ERROR Flag
ST(3-3)	: SBUSY Flag
ST(8-5)	: Interrupt mask for WSU1
ST(12-9)	: Interrupt mask for WSU2
ST(16-13)	: Interrupt mask for EMU

CARRY and ZERO flags are generated by ALU and they have their usual meaning. ERROR flag is set when previous instruction could not get executed and it resulted in error. For example, access to registers, on which a functional unit is operating, results in error. Those registers can only be accessed when functional unit is idle or in suspended state. Similarly, executing the instruction for transmitting data to the scheduler also results in error if SendToSch unit is busy transmiting the previous data. Status of the SendToSch unit can be checked with SBUSY Flag. This flag is set when the SendToSch Unit goes busy. Interrupt masks control generation of interrupts by functional units. If an interrupt is not masked in ST register, functional units will be generating that interrupt. For each functional unit, there are four bit masks corresponding to four different interrupts.

5.3 Instruction Set

Instructions set can be classified into four categories -

1. ALU Operations
2. Data Transfer Operations within PU
3. Subroutine and Interrupt Processing
4. Functional Unit Operations

Following notations have been used for describing instructions -

PR1, PR2	: Represent Program Unit Register
FR	: Represents Functional Unit Register
ADR	: Represents address of a memory
UNT	: Represents Functional Unit (i.e. WSU1, WSU2, EMU)
VALUE	: Represents 32-bit Immediate Value
[PR1][PR2]	: Content of PR1 or PR2 is treated as address

5.3.1 ALU Operations

Following instructions perform ALU operations -

ADD,	SUB,	AND,	EXOR,	OR,	NOT*,	SHIFTL*,	SHIFTR*
CADD,	CSUB,	CAND,	CEXOR,	COR,	CNOT*,	CSHIFTL*,	CSHIFTR*
ZADD,	ZSUB,	ZAND,	ZEXOR,	ZOR,	ZNOT*,	ZSHIFTL*,	ZSHIFTR*

Operations performed by them have their usual meanings. Mnemonic starting with the letter 'C' perform operation only if CARRY flag is set. Similarly, mnemonic starting with the letter 'Z' perform operation only if ZERO flag is set. Instruction marked with * take one argument while others take two. For example, instruction ADD PR1, PR2 adds content of registers PR1 and PR2 and puts the result in PR1. Similarly, instruction NOT PR1 complements the content of register PR1 and puts the result in same.

5.3.2 Data Transfer Operations within PU

Following instructions are there in this category -

MVDTOR PR1, ADR	: It moves the content of ADR into PR1
MVDTOM ADR, PR1	: It moves the content of PR1 to memory ADR
MVI PR1, VALUE	: Moves the VALUE into PR1
MVITOR PR1,[PR2]	: It moves the content of memory location whose address is in PR2 into PR1
MVITOM [PR1],PR2	: It moves the content of PR2 into memory location whose address is in PR1
MOVSCH PR1	: Transfers the content of PR1 to scheduler.

5.3.3 Subroutine and Interrupt Processing

This category has three instructions CALL, RET and ISERVCD. Format of CALL instruction is CALL ADR and it makes a call to subroutine starting from ADR. RET returns from subroutine call. ISERVCD instruction is executed to accept pending interrupts. While an interrupt is being processed, INTRcu does not doesn't generate further interrupts. Interrupts in between a service routine are registered and kept pending. Execution of ISERVCD instruction signals INTRcu that current interrupt processing is over and next interrupt will be accepted.

5.3.4 Functional Unit Instructions

There are three instructions in this category -

```
MVFTOR  PR1, FR
MVFTOF  FR, PR1
GO      UNT
```

MVFTOR transfers content of FR to PR1. MVFTOF transfers content of PR1 to FR. GO instruction is executed in an interrupt service routine. It's execution signals a functional unit to continue it's execution. Functional unit should have suspended it's operation earlier by interrupting PU.

5.4 Summary

Present chapter has described the DISEMP design. DISEMP has been designed with flexibility considerations. Program unit and functional units are loosely coupled through interrupts and micro-instructions. This approach of design introduces three levels in design -

1. Level at which data can be processed without PU intervention
2. Level at which data can be processed by only modifying MI register
3. Level at which data can be processed by modifying MI and other functional unit registers

Present chapter has described request processing briefly. Next chapter covers it in detail. DISEMP can, also, be used to perform complicated operations. Methods, to perform complicated operations at different level, are discussed in the next chapter.

Chapter 6

REQUEST PROCESSING

DISEMP accepts two types of requests - search request and match request. Each type can be further classified into Predefined and User defined request. Predefined requests are processed by functional units alone. User defined requests are interpreted by program unit. In this chapter, steps to process different types of request have been discussed. DISEMP can be used to perform complicated operations such as substring match. This chapter, also, describes the method of performing complicated operations.

6.1 Sequential Word Search Operation

Sequential word search operation, in WSU (Word Search Unit), searches a word sequentially in the dictionary partition. It is carried out in following steps -

1. Wait till a request comes in FS or SS registers. If it is a user defined request, WSU interrupts PU (Program Unit). If it is a predefined request and PU has requested for interrupt at this point, WSU interrupts PU. After interrupting, WSU waits for the continue signal. WSU receives continue signal when PU executes GO instruction.

2. The partition information, from partition table, is loaded into WP registers. Address of a partition entry is obtained by attaching 0 at the end of partition

number. WP0 register now contains the starting address of the partition and WP1 contains the record length and number of records in the partition.

3. A dictionary word is fetched from the dictionary partition. Five words are read starting from the address in WP0. The first word contains the address and length of information associated with the dictionary word and it is kept in WP2 register. Other four words contain a dictionary word which are read into FW registers. WSU checks if PU has requested to receive interrupt at this point. If so, it interrupts PU and waits for continue signal from PU.

4. WSU executes micro-instruction. At this step, fetched word and input word are compared. Result of comparison is put in the ST register. It further checks if PU is to be interrupted.

5. If result of comparison is TRUE, WSU goes into the state of transmitting result. It passes the request header to PKTFORM unit. MDTfsu is provided with the address and length of information from WP2 register. MDTfsu retrieves information associated with the dictionary word and passes it to the PKTFORM unit.

6. Number of records, in WP1 register, is decremented by one. Address of next word is calculated by incrementing the address in WP0 register by record length. WP0 register, now, contains the address of next word. If number of records in WP1 is non-zero, WSU goes to step 3.

7. WSU stops the search as soon as the number of records becomes zero. It further checks if PU has requested to receive interrupt at this point.

6.2 Binary Word Search Operation

Following are the steps, performed by binary word search operation -

1. Wait till a request comes into FS or SS registers. For a user defined request, it interrupts PU. If the request is predefined and PU has requested for an

interrupt at this point, WSU interrupts PU. Following steps have been described with the assumption that request has come into the FS registers. The description for SS register is very similar.

2. WSU requests for access to RI Bus. It transmits the input word, i.e., content of FS1 to FS4 registers, to cache. If word exists in the cache, it responds with the length and address of information which is associated with the word. In this case, WSU directly goes for the transmission of result.

3. Partition information is loaded into WP0 and WP1 registers. For loading the partition information, it follows the same steps which have been described in step 2 of the previous section.

4. WSU computes the address of the word in middle of the partition. For this, the number of records field in WP1 register, is shifted right by one position. Address of the middle word is obtained by multiplying this number with record length and adding it to the starting address in WP0 register. In other words, following computations have been performed to obtain the address of the middle word

$$Starting\ Address + Record\ Length * Number\ of\ records/2$$

Multiplication, in above computation, can be reduced to shifting by fixing record length to 8, 16 or 32 words.

5. The middle word, in the partition, is read into FW registers. After the word has been read, WP2 register contains the address and length of information associated with the word, and FW register set contains the dictionary word. WSU, further, looks for the interrupt to PU.

6. WSU executes micro-instruction. At this step, fetched word and input word are compared. Micro-instruction should perform *less than* operation in binary search. At the end of this step, WSU checks for the interrupt to PU.

7. If number of records field in WP1 register is 1, WSU goes to step 9.

8. Micro-instruction results in TRUE if the input word is less than the fetched word. In this case, the input word is searched in the first half of the partition otherwise in the second half. If the word is searched in the second half, WP0 register is loaded with the address from where the last word was fetched. Number of records field in WP1 register is incremented by 1. WSU goes to step 4.

9. WSU executes this step when number of records in the partition will be at the most 3. WSU reads these three records sequentially and performs comparison. If any of the three words match, input word is written into the cache and result is transmitted.

10. At the end of the search, WSU checks for the interrupt to PU.

6.3 Example Matching operation

The example matching operation is carried out in following steps -

1. Wait for a request to come in FM or SM registers. If it is a user defined request, EMU interrupts PU. For a predefined request, PU is interrupted if it requested for it. Following steps have been described with the assumption that request has come into the FM registers.

2. The partition information is loaded in EP registers. EP0 contains the starting address of the partition. EP1 contains the record length and number of records in example partition.

3. EMU loads the weights and attributes in WT and AT registers. The starting address in EP0 is the address at which first example reside. Address of weights and attributes are obtained by resetting lower five bits of address in EP0.

4. EMU fetches an example from the example partition. In this step, five words are read from the address contained in EP0. The first of these words contains the address and length of the information associated with example pattern.

and it is put in EP2 register. Other four words contain example pattern, and they are put in FE registers.

5. EMU executes the micro-instruction. At this step, distance is calculated between fetched example and input sentence pattern. At the end of this step, EMU checks if PU is to be interrupted.

6. If calculated distance is less than the previously calculated distance, EMU stores presently calculated distance. In this step, MD0 register is loaded with the presently calculated distance. Address and length of the information, associated with the present example, is transferred from EP2 register to MD1 register.

7. Number of records field in EP1 register is decremented by one. Address of next example is calculated by incrementing the address in the EP0 register by record length. If the number of records field in EP1 register is non-zero, EMU goes to step 4.

8. If the number of records field in EP1 register is zero, EMU stops matching. It passes the request header in FM0 register to PKTFORM unit. It transfers the address and length of the information to MDTfsu. Address and length are contained in MD1 register. MDTfsu retrieves the information associated with this example and passes it to the PKTFORM unit. Finally, EMU checks if PU is to be interrupted.

6.4 User Defined Operation

User defined packet is interpreted by program unit. On the arrival of this type of packet, functional unit interrupts PU. Interrupt service routine (ISR) is executed to examine packet. ISR takes one or more of following actions to start processing -

1. Make interrupt location point to the interrupt service routine

2. Load MI register

3. Setup interrupt mask register

For example, to retreive information of all those words whose length is given as l, DISEMP is sent a request containing (1) Packet Type and (2) Length l. On the arrival of request, ISR examines the type field and copies the length field into the memory. It modifies ST register to receive interrupt after word fetch. Interrupt location is setup for this interrupt. After every word fetch, PU is interrupted. ISR copies the fetched word into the general purpose registers of PU. It computes the length of the word and compares it with the given length l. If two lengths are same, ST register of WSU is modified to take it to the result transmission state. Otherwise, WSU is taken to the fetch word state. Another word is fetched and the same operation is repeated.

6.5 Summary

In present chapter, request processing, in DISEMP, has been described. Predefined requests are processed by functional units alone. User defined requests are interpreted by PU. PU interacts with functional units using interrupts and microinstructions. Program unit and functinal units cooperate to process user defined requests. DISEMP design has been simulated in Verilog. Implementation details are described in the next chapter.

Chapter 7

IMPLEMENTATION AND RESULT

DISEMP is designed for machine translation (MT) applications. It performs dictionary search and example matching which are basic operations for MT applications. The present work has been carried out in three phases -

1. Performance Measurement of a MT Software
2. Processor Design and Simulation
3. Design Verification

In the first phase, a MT software was studied to identify time consuming operations in translation. It is found that dictionary search and example matching take most of the time. In the second phase, DISEMP was designed and simulated in Verilog. In the third phase, DISEMP design and MT software were integrated to verify the design.

7.1 Performance Measurement of a MT Software

In this phase, a MT application was studied to identify time consuming operations and bottleneck areas. The MT software was developed by R. Jain et. al. at

Document Size	Number of Distinct Words
500	247
1000	388
1500	510
2000	618
2500	752
3000	840
3500	900
4000	973
4500	1036
5000	1090

Table 7.1: Distinct words in document

IIT, Kanpur. It translates from Hindi to English and uses the HEBMT approach for translation. The software has three major modules corresponding to three phases - Morphological Analysis, Identification of Syntactic Category and Example Matching. Unix utility *gprof* was used to study the software. The software was studied with dictionary base size of 700 words and example base size of 340 example patterns. Following conclusions were drawn after measuring performance of the software -

- Out of the three phases of translation, morph analysis (MA) and example matching take most of the time. While Identification of Syntactic Category (ISG) takes comparatively insignificant time. The MA and EM phase together take 98% of translation time.
- Percentage of translation time for the MA and EM phase varies drastically depending on the number of comparison and matching operations. Variation in percentage of time for EM phase was from 10% to 60% of translation time.
- Dictionary search takes most of the time of MA phase. It was found that it takes 60% of morph analysis.
- Distance calculation is a major operation in example matching. It takes about 80% of EM phase.

Since, most of the time goes in dictionary search and distance calculation, specific hardware for them will speed up the translation process. While carrying out the study, words were assumed to be present in memory. This assumption is valid as only few words are actively required during translation. As the size of document increases, number of distinct words don't increase as rapidaly (Table 7.1).

7.2 Processor Simulation

The results, presented in the previous section, indicate that the dictionary search and distance calculation take most of the time of translation. In second phase, a processor (DISEMP) was designed for them to obtain speedup in translation. DISEMP design was simulated in Verilog (Appendix A). List of modules and their interfaces are described in Appendix G.

As mentioned earlier, DISEMP has three external buses - EREQ Bus, ERES Bus and EM Bus. During design simulation, DISEMP was connected with a scheduler, a receiver and a memory module. Connection was made through interface pins which are described in Appendix B. Internal units within DISEMP were connected through four major buses - PC Bus, DF Bus, RI Bus and INTR Bus. The signals, for these buses, are described in Appendix C, D, E and F. Behavioral descriptions, for all these modules, were written in Verilog. In this section, few examples have been taken to describe the way different modules interact. Following discussion has been made around various signals whose descriptions are given in appendices.

Scheduler uses EREQ Bus to transmit the request. It first activates the signals *w_req* or *e_req* depending on the type of request. Internally, DISEMP checks if appropriate unit has free buffer and the RI Bus is free. If this condition satisfies, DISEMP signals scheduler by activating the signal *req_trnsmt*. On receipt of this signal, scheduler deactivates *w_req* or *e_req*. It puts the first byte of request on EREQ bus and activates *rqdt_ready*. DISEMP reads the first byte and activates the *req_trnsmt* signal. This operation is repeated 40 times to transmit the request.

As a next example, example fetch operation through MDTfsu is discussed. To retrieve an example from memory, EMU first raises the signal *TRNSMT_REQdf*.

MDTfsu services the requests in round robin fashion. When turn for EMU comes, MDTfsu activates *REQ_GRANTdf*. EMU then places the address and length of example on DF bus. Length is specified in number of machine words. Data width of DF bus is 32 bit. Address takes 24 bit and rest of the eight bits are used for length. MDTfsu retrieves each word from the memory and transmits it to the EMU. To transmit, it places the word on DATAdf and activates *RECV_DATAdf*. On receipt of *RECV_DATAdf* signal, EMU loads the FE registers with content of DATAdf. As soon as all the words of example has been fetched, MDTfsu goes to service next request.

Above examples have been taken just to illustrate the way DISEMP has been modelled. There are many other such operations which will be difficult to explain in this thesis.

7.3 Design Verification

To verify the design, it was integrated with the MT software. DISEMP memory was containing three word partitions and one example partition. Three word partitions were for noun, pronoun and verb where each one was containing around 20 words. Example partition was containing seven example patterns. During MA phase of translation, MT software was transmitting the request to DISEMP through a file. After searching the word, DISEMP keeps the result (i.e. the characteristics of the word) in a file which is read by MT software. Same steps repeat during EM phase. Input sentence pattern is kept in a file by the MT software. DISEMP calculates distances and passes the target pattern, associated with the example which has least distance, to MT software. The MT software performs translation based on the target pattern.

7.4 Comments on Performance

As mentioned earlier, DISEMP can also be used in a parallel environment. A parallel architecture, around DISEMP, has been described in chapter 3. The architecture

was designed to determine the services of DISEMP. The services of sentence memory unit (SMU) are distributed between the scheduler and DISEMP. The architecture exploits parallelism at three levels - (1) Translating multiple sentences in parallel (2) Translating a sentence in parallel, and (3) Performing dictionary search and example matching in parallel. In a parallel architecture, performance degrades due to communication overheads. First and second level of parallelism in the architecture are independent within themselves. Hence, communication overheads will be very less. Communication at second level of parallelism will depend on the way translation is carried out. Overheads at this level can be minimised by making job partitioning simple and keeping number of sentence processors low.

However, performance, in the architecture, may degrade due to speed mismatch. In this case, queue may get built up at various points such as scheduler, DISEMP etc. A simulation was planned by connecting multiple DISEMPs and a scheduler together to analyse queue formation. It could not be carried out as the license for Verilog expired.

7.5 Summary

Present thesis presented design of a processor (DISEMP) for MT applications. The work was carried out in three phases. In the first phase, dictionary search and example matching were found to be time consuming operations in translation. In the second phase, DISEMP was designed (for dictionary search and example matching operations) and simulated. During third phase, DISEMP design was integrated with the MT software to verify that it can be used for MT applications.

Chapter 8

CONCLUSION

Translation time, in a MT application, can be reduced by having specific hardware. In HEBMT approach, translation is carried out in three steps - (1) Morphological Analysis (2) Identification of Syntactic Category and (3) Example Matching. Dictionary search and example matching are basic operations for MT application and they take major portion of translation time. In present thesis, design of a processor (DISEMP) for dictionary search and example matching, has been described.

8.1 Features of DISEMP

DISEMP accepts a word or a sentence pattern as input. Dictionary base consists of words and their characteristics. Similarly, example base consists of examples and their target language patterns. In a search request, characteristics of those words, which satisfy search criteria, are transmitted as the result. In an example matching operation, distances, from a given sentence pattern, are calculated for all examples, and the target pattern of the example, which has least distance, is transmitted as result. DISEMP has two Word Search Units (WSU) and one Example Matching Unit (EMU).

Some performance enhancement features have also been incorporated in DISEMP. It supports partitioning of the database to narrow down search and match space. It has a cache to keep most frequently used words. DISEMP can be used

in a parallel environment. Before DISEMP, a parallel architecture was designed to consider issues related to parallelism.

Flexibility has been an important consideration during DISEMP designing. DISEMP can perform complicated search and match operations such as substring match, length based criteria etc. Flexibility has been incorporated in such a way that performance degradation doesn't occur due to instruction overheads.

8.2 Flexibility

Flexibility, in DISEMP, has been incorporated through program unit (PU). PU supports instructions which are quite basic in nature. Due to this, PU can perform complicated operations. Some of the complicated operations are listed below -

1. Retrieving characteristics of all those words whose length is as given
2. Retrieving charactristics of all those words which contain the given substring
3. Calculating the distance by ignoring some of the attributes

Performance degradation has been avoided by making the coupling loose between functional units and program unit. Most frequently used operations are performed by functional units alone and, hence, these operations don't suffer from instruction overheads. For least frequently used and complicated operations, program unit and functional units interact using following -

- Interrupts
- Micro-Instructions

This approach distributes the functionalities at three levels -

1. Level at which data can be processed by functional units alone.
2. Level at which data can be processed by modifying MI register only.
3. Level at which data can be processed by modifying MI and data registers both.

In DISEMP, most frequently used operations are implemented at first level which is fastest among the three. Other operations are implemented at second and third levels depending on their usage. The second Level is faster than the third level.

8.3 Implementation

The work, in this thesis, was carried out in three phases - 1. Performance Measurement of a MT Software, 2. Design and simulation of DISEMP, and 3.Design Verification. In first phase, it was found that dictionary search and example matching take major portion of the translation time. In second phase, DISEMP was designed for dictionary search and example matching operations to speed them up. DISEMP design was simulated in Verilog. In third phase, the DISEMP design was verified by integrating it with the MT software. The MT software used the services of DISEMP for dictionary search and example matching, and produced correct translations.

8.4 Future Work

Operations, in DISEMP, can be performed at three levels. The first level is fastest because there are no instruction overheads at this level. While, operations at the second and the third level take time because of PU involvement. To reduce overheads at second and third level, future work may be carried out in following directions -

1. Sophisticating program unit so that instruction execution is faster
2. Devicing a better scheme to reduce degree of communication between program unit and functional units.

Appendix A

AN EXAMPLE SESSION

In this appendix, an example session with simulation software is described. It explains the integration of DISEMP design with MT software. During simulation, DISEMP was containing one example partition and three dictionary partition. In the first section of this appendix, an example partition is given. Subsequently, interaction with software is described.

A.1 A Sample Example Partition

Following example partition contains three examples. Exaples are delimited by *. Each example has three syntactic units where each unit is given on one line. A syntactic unit is containing four attributes and one semantic tag. While the verb syntactic unit is containing only ony attribute, i.e., if it is transitive or intransitive. Example description is preceded by attribute types. Attribute types are status, gender, number, person and semantic tag. For the first example, attribute information is interpreted as - status as animate, gender as don't care, number as second and person as first.

```
---------------- Attribute Types
SGNP123
SGNP123
V
```

```
----------------  Examples Partition Starting
AXST211
INSX121
I
*
AXPF332
INPX112
I
*
INSX111
INPX112
R
*
1 V {to} 2      /* Target Pattern for First  Example */
1 V {for} 2     /* Target Pattern for Second Example */
2 V 1           /* Target Pattern for Third  Example */
*
```

A.2 A Translation Session

This session demonstrates the integration of MT software with DISEMP design. The MT software(*anu*) is executed which transmits the search and match request to DISEMP. Following is an example session -

```
csesun1:/1d2/warsi/project/mt/intg/example\_base$ anu
Please enter the sentence : vaha Gara jA rahA hE.
/* Verb analysis being performed by software */
------------------    STARTING  MORPH   ANALYSIS  ------------
Activate processor to search  the word [vaha Gara].Enter When search is over.
/* DISEMP now searching the word [vaha Gara] */
Word has not been found
```

```
Activate processor to search  the word [vaha].Enter When search is over.
Word has not been found
Activate processor to search  the word [vaha].Enter When search is over.
Word has been found
Activate processor to search  the word [Gara].Enter When search is over.
Word has been found
------------  Initiating Identification of Syntactic Group ---------
-----------------  INITIATING  EXAMPLE  MATCHING  PHASE --------
Activate Processor to perfom Example Matching. Press  CR  when over
/* DISEMP now performing example matching */
Translated Sentence [1] is  --
he is going to house
```

A.3 A DISEMP Simulation Session

This session explains DISEMP design simulation. Following is a portion of Verilog simulation output for an example matching operation. Each line first carries module name, where activity is taking place, and the time (within bracket) of action. Next to the module name, is the action taken by it.

```
Highest level modules:
SYSbu

MDTfsu(80):PU_RDREQ
RIBUSbau(82) : Granting RIBUS to RQTFSU Example
INSTRcu(171): Instr Fetched [MVI CNTRLU 01, 00ffffd0  ] ADDRESS fffffc
EMU(191) : Received Request 852304a0
EMU(291) : Received Request 3ca45844
MDTfsu(340):PU_RDREQ
EMU(391) : Received Request e4900000
INSTRcu(431): Instr Fetched [MVI REGFILE1U 07, 00000000  ] ADDRESS ffffd0
```

```
EMU(491) : Received Request 00000000
EMU(591) : Received Request 00000000
MDTfsu(600):PU_RDREQ
EMU(660) : Loading Partition Info
INSTRcu(691): Instr Fetched [MVINTOR REGFILE1U 08,  ] ADDRESS ffffd2
MDTfsu(760):EMU_RDREQ
EMU(811) : Loading Partition Info PRTN0 00000972
EMU(851) : Loading Partition Info PRTN1 05000005
MDTfsu(880):PU_RDREQ
MDTfsu(1000):EMU_RDREQ
EMU(1760) : Fetching Example
MDTfsu(1760):PU_RDREQ
INSTRcu(1851): Instr Fetched [ADD REGFILE1U 08,  ] ADDRESS ffffd3
MDTfsu(1920):EMU_RDREQ
EMU(1971) : Received Exampl 0400098b
EMU(2011) : Received Exampl 3ca44844
EMU(2051) : Received Exampl e5100000
EMU(2091) : Received Exampl 00000000
EMU(2131) : Received Exampl 00000000
MDTfsu(2160):PU_RDREQ
EMU(2180) : Executing Micro Instruction
INSTRcu(2251): Instr Fetched [MVI REGFILE1U 05, 7fffffff  ] ADDRESS ffffd4
MDTfsu(2420):PU_RDREQ
INSTRcu(2511): Instr Fetched [SUB REGFILE1U 05,  ] ADDRESS ffffd6
MDTfsu(2680):PU_RDREQ
INSTRcu(2771): Instr Fetched [MVDTOM REGFILE1U 08, 000000  ] ADDRESS ffffd7
MDTfsu(2860):PU_WRREQ
MDTfsu(3000):PU_RDREQ
INSTRcu(3091): Instr Fetched [MVDTOR REGFILE1U 06, ffffa0  ] ADDRESS ffffd9
MDTfsu(3180):PU_RDREQ
MDTfsu(3400):PU_RDREQ
```

```
INSTRcu(3491): Instr Fetched [MVDTOM REGFILE1U 06, ffff90  ] ADDRESS ffffdb
MDTfsu(3580):PU_WRREQ
MDTfsu(3720):PU_RDREQ
INSTRcu(3811): Instr Fetched [MVI REGFILE1U 0b, 00fffffc  ] ADDRESS ffffdd
MDTfsu(3980):PU_RDREQ
INSTRcu(4071): Instr Fetched [MVI REGFILE1U 03, 000001f4  ] ADDRESS ffffdf
MDTfsu(4240):PU_RDREQ
INSTRcu(4331): Instr Fetched [CALL ffffe8  ] ADDRESS ffffe1
MDTfsu(4500):PU_WRREQ
EMU(4620) : Computing Minimum Distance
EMU(4621) : MIN_INFO : 0400098b   MIN_DIST : 00000004
MDTfsu(4640):PU_RDREQ
EMU(4660) : Computing Next Address
EMU(4680) : Fetching Example
INSTRcu(4731): Instr Fetched [ADD REGFILE1U 09,  ] ADDRESS ffffe8
MDTfsu(4800):EMU_RDREQ
EMU(4851) : Received Exampl 0300098f
EMU(4891) : Received Exampl 3c06d040
EMU(4931) : Received Exampl e4a00000
EMU(4971) : Received Exampl 00000000
EMU(5011) : Received Exampl 00000000
        .
        .
EMU(16120) : Computing Minimum Distance
EMU(16121) : MIN_INFO : 02000997   MIN_DIST : 00000001
EMU(16160) : Computing Next Address
EMU(16200) : Transmitting Result
INSTRcu(16211): Instr Fetched [MVI CNTRLU 01, 00fffffc  ] ADDRESS ffffe2
MDTfsu(16280):EMU_RSREQ
MDTfsu(16400):PU_RDREQ
INSTRcu(16491): Instr Fetched [MVI CNTRLU 01, 00fffffd0  ] ADDRESS fffffc
```

```
MDTfsu(16660):PU_RDREQ
INSTRcu(16751): Instr Fetched [MVI REGFILE1U 07, 00000000  ] ADDRESS ffffd0
MDTfsu(16920):PU_RDREQ
L403 "PROC.v": $finish at simulation time 250000
530031 simulation events + 106 accelerated events
CPU time: 1.6 secs to compile + 2.1 secs to link + 4.7 secs in simulation
End of VERILOG-XL 2.2.1   Dec 12, 1996  15:39:08
```

Appendix B

PROCESSOR INTERFACE SIGNALS

Signal	Width	Description
EM Bus Interface Signals		
mdt_bus	32	Data Bus
ma_bus	24	Address Bus
mdt_enable	1	Memory Enable
mdt_rw	1	Memory Read/Write
mdt_ready	1	Data Ready
EREQ Bus Interface Signals		
rqdt_bus	8	EREQ Data Bus
w_req	1	Receive Word Request
e_req	1	Receive Example Request
req_trnsmt	1	DISEMP ready to receive request
rqdt_schrecv	1	Scheduler to receive data
rqdt_ready	1	Scheduler ready for next data
rqdt_bus_req	1	Request for access to EREQ Bus
rqdt_bus_grant	1	EREQ Bus Granted
rqdt_bus_release	1	EREQ Bus released

Signal	Width	Description
ERES Bus Interface Signals		
rsdt_bus	8	ERES Data Bus
rsdt_recv	1	RECV to receive data
rsdt_ready	1	RECV has received data
rsdt_bus_req	1	Request for access to ERES bus
rsdt_bus_grant	1	ERES Bus Granted to DISEMP
rsdt_bus_release	1	ERES Bus released by DISEMP

Appendix C

PC BUS SIGNALS

Signal	Width	Description
DATApc	32	Carries Data
SRC_REGpc	6	Source Register Number
DST_REGpc	6	Destination Register Number
PC_RWpc	1	Read/Write Signal within PU
FNC_RWpc	1	Read/Write for functional unit
ALU_OPRpc	3	ALU Operation
EN_ALUpc	1	ALU Enable
EN_REGFILE1pc	1	Register File Enable(first)
EN_REGFILE2pc	1	Register File Enable(second)
EN_INTRpc	1	INTRcu Enable
EN_SSCHpc	1	SendToSch Enable
EN_WSU1pc	1	WSU1 Enable
EN_WSU2pc	1	WSU2 Enable
EN_EMUpc	1	EMU Enable
EN_STREGpc	1	ST Register Enable
CONT_WSU1pc	1	Continue Signal for WSU1
CONT_WSU2pc	1	Continue Signal for WSU2
CONT_EMUpc	1	Continue Signal for EMU

Signal	Width	Description
CARRYpc	1	CARRY Flag
ZEROpc	1	ZERO Flag
SBUSYpc	1	SendToSch Unit Busy Flag
ERRORpc	1	ERROR Flag
SET_ERRORpc	1	Set ERROR Flag
CLR_ERRORpc	1	Clear ERROR Flag
SET_SBUSYpc	1	Set SBUSY Flag
CLR_SBUSYpc	1	Clear SBUSY Flag
SET_CARRYpc	1	Set CARRY Flag
CLR_CARRYpc	1	Clear CARRY Flag
SET_ZEROpc	1	Set ZERO Flag
CLR_ZEROpc	1	Clear ZERO Flag
INTR_ARRpc	1	Interrupt Generated by INTRcu
INTR_ACKpc	1	Interrupt Acknowledged by INSTRcu
INTR_SERVICEDpc	1	Interrupt Serviced

Appendix D

DF BUS SIGNALS

Signal	Width	Description
DATAdf	32	Carries Data
TRNSMT_REQdf	4	Request to access memory
RESULT_REQdf	3	Request to transmit result
REQ_GRANTdf	4	Request Granted
RECV_DATAdf	1	Receive Data for earlier request
OVERdf	1	Information accessed for result
RWdf	1	Read/Write
PKTUNIT_BUSYdf	1	PKTFORM Unit Busy
EN_PUdf	1	Enable PU
EN_WSU1df	1	Enable WSU1
EN_WSU2df	1	Enable WSU2
EN_EMUdf	1	Enable EMU
EN_PKTUNITdf	1	Enable PKTFORM Unit

Appendix E

RI BUS SIGNALS

Signal	Width	Description
DATAri	32	Carries Data
PU_REQri	1	Request to access bus from PU
WSU1_REQri	1	Request to access bus from WSU1
WSU1_CACHEri	1	Cache Read/Write from WSU1
WSU2_REQri	1	Request to access bus from WSU2
WSU2_CACHEri	1	Cache Read/Write from WSU2
RQTFSU_REQWri	1	Request to transmit word request from RQTfu
RQTFSU_REQEri	1	Request to transmit example request from RQTfu
PU_GRANTri	1	Request granted to PU
WSU1_GRANTri	1	Request granted to WSU1
WSU2_GRANTri	1	Request granted to WSU2
RQTFSU_GRANTWri	1	Request granted to RQTfu for word request
RQTFSU_GRANTEri	1	Request granted to RQTfu for example request
BUS_RELEASEri	1	Bus released signal
EN_WSU1ri	1	Enable WSU1
EN_WSU2ri	1	Enable WSU2
EN_EMUri	1	Enable EMU
EN_CACHEri	1	Enable Cache
EN_RQTFSUri	1	Enable RQTfu

Signal	Width	Description
CACHE_RWri	1	Operation being performed on cache
RECV_DATAri	1	Receive Data for word or exmple request
RQTFSU_FREEri	1	RQTfu free for sending data to scheduler
WSU1_FREEri	1	WSU1 Free to receive a request
WSU2_FREEri	1	WSU2 Free to receive a request
EMU_FREEri	1	EMU Free to receive a request
CACHE_MISSri	1	If word was found in cache

Appendix F

INTR BUS SIGNALS

Signal	Width	Description
REQWSU1N1intr	1	Interrupt from WSU1 of Type 1
REQWSU1N2intr	1	Interrupt from WSU1 of Type 2
REQWSU1N3intr	1	Interrupt from WSU1 of Type 3
REQWSU1N4intr	1	Interrupt from WSU1 of Type 4
REQWSU2N1intr	1	Interrupt from WSU2 of Type 1
REQWSU2N2intr	1	Interrupt from WSU2 of Type 2
REQWSU2N3intr	1	Interrupt from WSU2 of Type 3
REQWSU2N4intr	1	Interrupt from WSU2 of Type 4
REQEMUN1intr	1	Interrupt from EMU of Type 1
REQEMUN2intr	1	Interrupt from EMU of Type 2
REQEMUN3intr	1	Interrupt from EMU of Type 3
REQEMUN4intr	1	Interrupt from EMU of Type 4
MASKWSU1N1intr	1	Mask for Interrupt (WSU1, Type1)
MASKWSU1N2intr	1	Mask for Interrupt (WSU1, Type2)
MASKWSU1N3intr	1	Mask for Interrupt (WSU1, Type3)
MASKWSU1N4intr	1	Mask for Interrupt (WSU1, Type4)

Signal	Width	Description
MASKWSU2N1intr	1	Mask for Interrupt (WSU2, Type1)
MASKWSU2N2intr	1	Mask for Interrupt (WSU2, Type2)
MASKWSU2N3intr	1	Mask for Interrupt (WSU2, Type3)
MASKWSU2N4intr	1	Mask for Interrupt (WSU2, Type4)
MASKEMUN1intr	1	Mask for Interrupt (EMU, Type1)
MASKEMUN2intr	1	Mask for Interrupt (EMU, Type2)
MASKEMUN3intr	1	Mask for Interrupt (EMU, Type3)
MASKEMUN4intr	1	Mask for Interrupt (EMU, Type4)

Appendix G

MODULE INTERFACE

Modules and their interface signals have been described in this appendix. These modules are internal to the processor. Modules are connected to the four major buses, i.e., PC Bus, DF Bus, RI Bus and INTR Bus. Interface signals to these modules are the bus signals which have been explained earlier. First, list of modules within the processor is given alongwith their interface. In second section, an example Verilog code is given in which RQTfsu receives a request and transmits it to the functional unit.

G.1 List of Modules

MDTfsu

---

mdt_bus,	ma_bus,	mdt_enable,
mdt_rw,	mdt_ready,	
DATAdf,	TRNSMT_REQdf,	RESULT_REQdf,
REQ_GRANTdf,	RECV_DATAdf,	OVERdf,
RWdf,	PKTUNIT_BUSYdf,	EN_PUdf,
EN_WSU1df,	EN_WSU2df,	EN_EMUdf,
EN_PKTUNITdf,	CLOCK	

RQTfsu

---

rqdt_bus,	w_req,	e_req,
req_trnsmt,	rqdt_schrecv,	rqdt_ready,
rqdt_bus_req,	rqdt_bus_grant,	rqdt_bus_release,
DATAri,	RQTFSU_REQWri,	RQTFSU_REQEri,
RQTFSU_GRANTWri,	RQTFSU_GRANTEri,	BUS_RELEASEri,
RECV_DATAri,	RQTFSU_FREEri,	EN_RQTFSUri,
CLOCK		

PKTFORMbu

---

rsdt_bus,	rsdt_recv,	rsdt_ready,
rsdt_bus_req,	rsdt_bus_grant,	rsdt_bus_release,
DATAdf,	REQ_GRANTdf[2:0],	
RESULT_REQdf,		
RECV_DATAdf,	OVERdf,	PKTUNIT_BUSYdf,
EN_PKTUNITdf,		
CLOCK		

ALUbu

DATApc,	DATA2si,	DATApc,
ALU_OPRpc,	EN_ALUpc,	SET_CARRYpc,
CLR_CARRYpc,	SET_ZEROpc,	CLR_ZEROpc,
CLOCK		

REG_FILEmem

DATApc,	SRC_REGpc,	PC_RWpc,
EN_REGFILE1pc,	DATA2si,	DST_REGpc,

EN_REGFILE2pc, CLOCK

STmem

DATApc, EN_STREGpc, PC_RWpc,
MASKNEintr, MASKWSU1N1intr, MASKWSU1N2intr,
MASKWSU1N3intr, MASKWSU1N4intr,
MASKWSU2N1intr, MASKWSU2N2intr, MASKWSU2N3intr,
MASKWSU2N4intr,
MASKEMUN1intr, MASKEMUN2intr, MASKEMUN3intr,
MASKEMUN4intr,
CARRYpc, ZEROpc, ERRORpc,
SBUSYpc,
SET_CARRYpc, CLR_CARRYpc,
SET_ZEROpc, CLR_ZEROpc,
SET_ERRORpc, CLR_ERRORpc,
SET_SBUSYpc, CLR_SBUSYpc,
CLOCK

INSTRcu

DATApc, SRC_REGpc, DST_REGpc,
PC_RWpc, FNC_RWpc, ALU_OPRpc,
EN_ALUpc, EN_REGFILE1pc, EN_REGFILE2pc,
EN_INTRpc, EN_SSCHpc, EN_WSU1pc,
EN_WSU2pc, EN_EMUpc, EN_STREGpc,
OVERpc, ERRORpc, CARRYpc,
ZEROpc, SBUSYpc,
SET_ERRORpc, CLR_ERRORpc, CONT_WSU1pc,
CONT_WSU2pc, CONT_EMUpc,
INTR_ARRpc, INTR_ACKpc, INTR_SERVICEDpc,

```
RI_TRNSMTpc,
DATAdf,                    TRNSMT_REQdf[3],
REQ_GRANTdf[3],
RWdf,                      RECV_DATAdf,        EN_PUdf,
CLOCK

RIBUSbau
-----------------------------------------------------------
PU_REQri,                  WSU1_REQri,         WSU2_REQri,
RQTFSU_REQWri,             RQTFSU_REQEri,      WSU1_CACHEri,
WSU2_CACHEri,
PU_GRANTri,                WSU1_GRANTri,       WSU2_GRANTri,
RQTFSU_GRANTWri,           RQTFSU_GRANTEri,
BUS_RELEASEri,
RQTFSU_FREEri,             WSU1_FREEri,        WSU2_FREEri,
EMU_FREEri,
EN_WSU1ri,                 EN_WSU2ri,          EN_EMUri,
EN_CACHEri,                EN_RQTFSUri,        CACHE_RWri,
CLOCK

SEND_TO_SCHbu
-----------------------------------------------------------
DATApc,                    EN_SSCHpc,          RI_TRNSMTpc,
SET_SBUSYpc,               CLR_SBUSYpc,
DATAri,                    PU_REQri,           PU_GRANTri,
BUS_RELEASEri,
CLOCK

INTRPTcu
-----------------------------------------------------------
DATApc,                    EN_INTRpc,
```

```
INTR_ARRpc,                 INTR_ACKpc,          INTR_SERVICEDpc,
MASKNEintr,                 MASKWSU1N1intr,      MASKWSU1N2intr,
MASKWSU1N3intr,             MASKWSU1N4intr,
MASKWSU2N1intr,             MASKWSU2N2intr,      MASKWSU2N3intr,
MASKWSU2N4intr,
MASKEMUN1intr,              MASKEMUN2intr,       MASKEMUN3intr,
MASKEMUN4intr,
REQNEintr,                  REQWSU1N1intr,       REQWSU1N2intr,
REQWSU1N3intr,              REQWSU1N4intr,
REQWSU2N1intr,              REQWSU2N2intr,       REQWSU2N3intr,
REQWSU2N4intr,
REQEMUN1intr,               REQEMUN2intr,        REQEMUN3intr,
REQEMUN4intr,
CLOCK

WSUbu
-------------------------------------------------------------
DATApc,                     DST_REGpc,           FNC_RWpc,
EN_WSU1pc,                  SET_ERRORpc,         CLR_ERRORpc,
CONT_WSU1pc,
REQWSU1N1intr,              REQWSU1N2intr,       REQWSU1N3intr,
REQWSU1N4intr,
MASKWSU1N1intr,             MASKWSU1N2intr,      MASKWSU1N3intr,
MASKWSU1N4intr,
DATAri,                     WSU1_REQri,          WSU1_GRANTri,
RECV_DATAri,                EN_WSU1ri,           WSU1_CACHEri,
WSU1_FREEri,                CACHE_MISSri,
BUS_RELEASEri,
DATAdf,                     TRNSMT_REQdf[0],
RESULT_REQdf[0],
REQ_GRANTdf[0],EN_WSU1df,   RECV_DATAdf,
```

```
CLOCK

CACHEbu
-------------------------------------------------------------
DATAri,                    EN_CACHEri,          CACHE_RWri,
CACHE_MISSri,              CLOCK

EMUbu
-------------------------------------------------------------
DATApc,                    DST_REGpc,           FNC_RWpc,
EN_EMUpc,                  SET_ERRORpc,         CLR_ERRORpc,
CONT_EMUpc,
REQEMUN1intr,              REQEMUN2intr,        REQEMUN3intr,
REQEMUN4intr,
MASKEMUN1intr,             MASKEMUN2intr,       MASKEMUN3intr,
MASKEMUN4intr,
DATAri,                    RECV_DATAri,         EN_EMUri,
EMU_FREEri,                DATAdf,              TRNSMT_REQdf[2],
RESULT_REQdf[2],           REQ_GRANTdf[2],      EN_EMUdf,
RECV_DATAdf,               CLOCK

CLOCKbu
-------------------------------------------------------------
CLOCK
```

G.2 Verilog Code for Receiving a Request

```
always @RECV_SETsi
begin
  // Recieve first byte
```

```
@(negedge CLK);
STATEsi='INACTIVEst;
FSU_FREEse = 'DE_ACTIVE;
RI_DATADse = 'DATABUS_WD'bz;
RECV_DATAse = 'DE_ACTIVE;

req_trnsmt='ACTIVE;
wait(rqdt_ready=='ACTIVE);
@(posedge CLK);
BUFFERsi[7:0]=rqdt_bus;
req_trnsmt='DE_ACTIVE;

// Recieve second byte
@(negedge CLK);
STATEsi='INACTIVEst;
req_trnsmt='ACTIVE;
wait(rqdt_ready=='ACTIVE);
@(posedge CLK);
BUFFERsi[15:8]=rqdt_bus;
req_trnsmt='DE_ACTIVE;

// Recieve third byte
@(negedge CLK);
STATEsi='INACTIVEst;
req_trnsmt='ACTIVE;
wait(rqdt_ready=='ACTIVE);
@(posedge CLK);
BUFFERsi[23:16]=rqdt_bus;
req_trnsmt='DE_ACTIVE;

// Recieve fourth byte
```

```
  @(negedge CLK);
  STATEsi='INACTIVEst;
  req_trnsmt='ACTIVE;
  wait(rqdt_ready=='ACTIVE);
  @(posedge CLK);
  BUFFERsi[31:24]=rqdt_bus;
  req_trnsmt='DE_ACTIVE;

  // Transmit data to functional unit
  @(negedge CLK);
  RECV_DATAse='ACTIVE;
  @(posedge CLK);
  RI_DATADse = BUFFERsi;
  -> SET_RECVDsi;
end
```

References

[Bur82] Forbes J. Burkowski. A Hardware Hashing Scheme in the Design of a Multiterm String Comparator. *IEEE Transactions on Computers*, September 1982.

[DL86] Wayne A. Davis and De-Lei Lee. Fast Search Algorithm for Associative Memories. *IEEE Transactions on Computers*, May 1986.

[Hen84] John L. Hennessy. VLSI Processor Architecture. *IEEE Transactions on Computers*, December 1984.

[H.K93a] H.Kitano. Challenges of Massive Parallelism. In *IJCAI-93*, 1993.

[H.K93b] H.Kitano. A Comprehensive and Practical Model of Memory-based Machine Translation. In *IJCAI-93*, 1993.

[HYT87] Hajime Nagai Hachiro Yamada, Masaki Hirata and Kousuke Takahashi. A High Speed String Search Engine. *IEEE J Solid State Circuits*, October 1987.

[Jai95] Renu Jain. *HEBMT : A Hybrid Example-Based Approach for Machine Translation.* PhD thesis, Birla Institute of Technology and Science, PILANI, 1995.

[JNK88] Shin-Ichiro Yamada Jiro Naganuma, Takeshi Ogura and Takashi Kimura. High Speed CAM Based Architecture for a Prolog Machine (ASCA). *IEEE Transactions on Computers*, November 1988.

[Lop84] Lanfranco Lopriore. Capability Based Tagged Architecture. *IEEE Transactions on Computers*, September 1984.

[MHT88] Hajime Nagai Masaki Hirata, Hachiro Yamada and Kousuke Takahashi. A Versatile Data String Search VLSI. *IEEE J Solid State Circuits*, April 1988.

[RJR95] Ajai Jain Renu Jain and R.M.K.Sinha. Emerging Trends in Machine Translation Between English and Indian Languages. Technical report, IIT, Kanpur, January 1995.

[SL91] David R. Smith and Jing C. Lin. The Tree Match Chip. *IEEE Transactions on Computers*, May 1991.

[Suk96] P. Sukumar. An ASIC Designed for NLP Applications. Master's thesis, IIT, Kanpur, 1996.

[TAOS82] Arnold L. Rosenberg Thomas A. Ottmann and Larry J. Stockmeyer. A Dictionary Machine for VLSI. *IEEE Transactions on Computers*, September 1982.